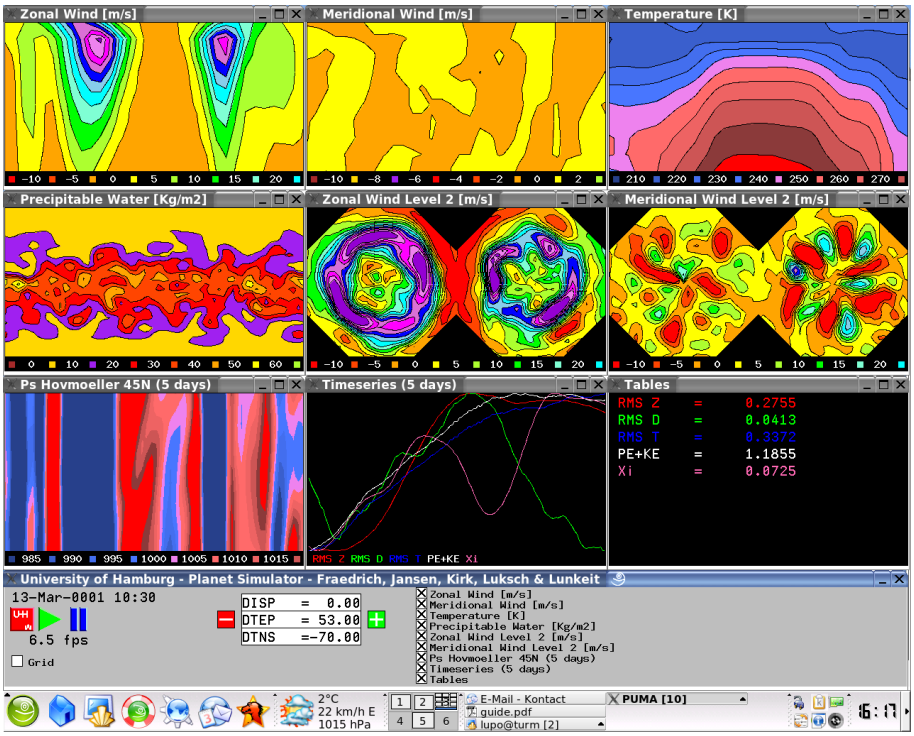
Planet Simulator



User's Guide

Version 1

F. Lunkeit S. Blessing K. Fraedrich H. Jansen E. Kirk
U. Luksch F. Sielmann

March 2, 2005

Contents

1 Installation **5**
- 1.1 Quick Installation . . . 5
- 1.2 Directory structure . . . 5
- 1.3 Model build phase . . . 6
- 1.4 Model run phase . . . 6

2 Modules **9**
- 2.1 fluxmod.f90 . . . 10
- 2.2 miscmod.f90 . . . 11
- 2.3 surfmod.f90 . . . 12
- 2.4 fftmod.f90 . . . 13
- 2.5 landmod.f90 . . . 14
- 2.6 legmod.f90 . . . 16
- 2.7 mpimod.f90 and mpimod_dummy.f90 . . . 17
- 2.8 outmod.f90 . . . 19
- 2.9 puma.f90 . . . 20
- 2.10 pumamod.f90 . . . 24
- 2.11 radmod.f90 . . . 25
- 2.12 rainmod.f90 . . . 27
- 2.13 seamod.f90 . . . 28
- 2.14 Files . . . 30
- 2.15 Sea ice and ocean modules . . . 31
  - 2.15.1 seamod.f90 . . . 35
  - 2.15.2 intermodatm.f90 . . . 36
  - 2.15.3 intermodice.f90 . . . 37
  - 2.15.4 icemod.f90 . . . 38
  - 2.15.5 oceanmod.f90 . . . 39
  - 2.15.6 oceanmod50.f90 . . . 40

3 Running Planet Simulator **41**
- 3.1 Interactive Console Mode . . . 41
- 3.2 Batch Mode . . . 42

4 Parallel Program Execution **49**
- 4.1 Concept . . . 49
- 4.2 Parallelization in Gridpoint Domain . . . 49

4.3 Parallelization in Spectral Domain . . . 50
4.4 Synchronization points . . . 50
4.5 Source code . . . 51

5 Graphical User Interface **53**
5.1 Graphical user interface (GUI) . . . 53

6 Postprocessor Pumaburner **57**
6.1 Introduction . . . 57
6.2 Usage . . . 57
6.3 Namelist . . . 58
6.4 HTYPE . . . 58
6.5 VTYPE . . . 58
6.6 MODLEV . . . 59
6.7 hPa . . . 59
6.8 MEAN . . . 59
6.9 Format of output data . . . 59
6.10 SERVICE format . . . 60
6.11 HHMM . . . 60
6.12 HEAD7 . . . 61
6.13 MARS . . . 61
6.14 MULTI . . . 61
6.15 Namelist example . . . 61
6.16 Troubleshooting . . . 62

7 Graphics **63**
7.1 Grads . . . 63
7.2 Vis5D . . . 66

Bibliography **67**

A List of Constants and Symbols **69**
A.0.1 List of Constants and Symbols . . . 70

B Puma Codes **75**

C Namelist **79**

Chapter 1

Installation

The whole package containing the models "Planet Simulator" and "PUMA" along with "Most", the "Model Starter" comes in a single file, usually named "Most(n).tar.gz" with (n) specifying a version number. The following subsection gives an example, assuming version 9.

1.1 Quick Installation

```
gunzip Most9.tar.gz
tar -xvf Most9.tar
cd Most9
make
most.x
```

If this sequence of commands produces error messages, consult the "FAQ" (Frequent Asked Questions) and README files in the Most9 directory. They are plain text files, that can be read with the command "more" or any text editor.

1.2 Directory structure

```
home/Most9> ls -lg

-rw-r--r--  1   820  FAQ           <- Frequent Ask Questions
-rw-r--r--  1   550  README        <- Actual information
-rwxr-xr-x  1    57  cleanplasim   <- command clears plasim directories
-rwxr-xr-x  1    51  cleanpuma     <- command clears puma directories
-rw-r--r--  1    83  makefile      <- Used to "make" most.x
-rw-r--r--  1 63763  most.c        <- Source for Most (Model Starter)
-rwxr-xr-x  1 66138  most.x        <- Most executable created by "make"
drwxr-xr-x  8   192  plasim        <- Planet Simulator directory
drwxr-xr-x  2   136  postprocessor <- Postprocessor directory
drwxr-xr-x  7   168  puma          <- PUMA directory
```

The directory structure must not be changed, even empty directories must be kept as they are, the Most program relies on the existence of these directories!

For each model, currently "Planet Simulator" and "PUMA" exist a directory with following subdirectories:

```
Most9/plasim> ls -lg

drwxr-xr-x  2   128  bin   <- model executables
drwxr-xr-x  2  1824  bld   <- build directory
drwxr-xr-x  2   280  dat   <- initial and boundary data
drwxr-xr-x  2    80  doc   <- documentation, user's guide, reference manual
drwxr-xr-x  2   928  run   <- run directory
drwxr-xr-x  2  1744  src   <- source code
```

After installation only "dat", "doc" and "src" contain files, all other directories are empty.

Running "Most" to configure a model and define an experiment uses the directories in the following manner:

1.3 Model build phase

Most writes an executable shell script to the "bld" directory and executes it directly hereafter. It copies all necessary source files from "src" to "bld" and modifies them according to the selected parameter configuration. Modification of source code is necessary for vertical and horizontal resolution change and for using more than 1 processor (parallel program execution). The original files in the "src" directory are not changed by Most. The program modules are then compiled and linked using the "make" command, also issued by Most. Most provides two different makefiles for the single CPU version or the parallel version (using MPI, Message Passing Interface). The resolution and CPU parameters are coded into the filename of the executable, in order to have different names for different versions. E.g. the executable "most_plasim_t21_l10_p2.x" is an executable compiled for a horizontal resolution of T21, a vertical resolution of 10 levels and 2 CPU's. The executable is copied to the model's "bin" directory after building. Each time Most is used to setup a new experiments it checks the "bin" directory for a matching executable. If it's there, it's used without rebuilding otherwise a new executable with the selected parameters is created. Rebuilding may be forced by using the cleanplasim command in the Most directory. The build directory is not cleared after usage. The user may want to modify the makefile or the build script for his own purposes and start the building directly by executing "most_plasim_build". For permanent user modifications the contents of the "bld" directories have to be copied elsewhere, because each usage of Most overwrites the contents of "bld".

1.4 Model run phase

After building the model with the selected configuration, Most writes or copies all necessary files to the model's "run" directory. These are the executable, initial and boundary data, namelist files containing the parameter and finally the run script itself. Depending on the exit from Most, either "Save & Exit" or "Run & Exit", the run script is started from Most and takes control of the model run. A checkmark on GUI invokes also the the Graphical User Interface for user

control and display of variables during the run. Again all contents of the "run" directory are subject of change for the user. But it would be wise to keep changed run setups in other, user created directories, because each usage of Most will overwrite the contents of the run directory. Another concept could be to rename user changed files, because Most always generated files starting with "most_" and leaves other files untouched.

Chapter 2

Modules

This is a technical documentation of the PUMA-II model. In the following, the purposes of the individual modules is given and the general structure and possible input and output opportunities (namelist, files) are explained.

2.1 fluxmod.f90

General The module fluxmod.f90 contains subroutines to compute the different surface fluxes and to perform the vertical diffusion. The interface to the main PUMA module puma.f90 is given by the subroutines *fluxini*, *fluxstep* and *fluxstop* which are called in puma.f90 from the subroutines *prolog*, *gridpointd* and *epilog*, respectively.

Input/Output fluxmod.f90 does not use any extra input file or output file and is controlled by the namelist *fluxpar* which is part of the namelist file **puma_namelist**:

Parameter	Type	Purpose	Default
NEVAP	Integer	Switch for surface evaporation (0 = off, 1 = on)	1
NSHFL	Integer	Switch for surface sensible heat flux (0 = off, 1 = on)	1
NSTRESS	Integer	Switch for surface wind stress (0 = off, 1 = on)	1
NTSA	Integer	Switch for computing the near surface air temperature which is used for the Richardson number (1 = potential temperature, 2 = virtual potential temperature)	2
NVDIFF	Integer	Switch for vertical diffusion (0 = off, 1 = on)	1
VDIFF_LAMM	Real	Tuning parameter for vertical diffusion	160.
VDIFF_B	Real	Tuning parameter for vertical diffusion	5.
VDIFF_C	Real	Tuning parameter for vertical diffusion	5.
VDIFF_D	Real	Tuning parameter for vertical diffusion	5.

Structure Internally, fluxmod.f90 uses the FORTRAN-90 module *fluxmod*, which uses the global common module *pumamod* from pumamod.f90. Subroutine *fluxini* reads the namelist and, if the parallel version is used, distributes the namelist parameters to the different processes. Subroutine *fluxstep* calls the subroutine *surflx* to compute the surface fluxes and calls the subroutine *vdiff* to do the vertical diffusion. Subroutine *fluxstop* is a dummy subroutine since there is nothing to do to finalize the computations in fluxmod.f90. The computation of the surface fluxes in *surflx* is spitted into several parts. After initializing the stability dependent transfer coefficients, the subroutines *mkstress*, *mkshfl* and *mkevap* do the computations which are related to the surface wind stress, the surface sensible heat flux and the surface evaporation, respectively.

2.2 miscmod.f90

General The module miscmod.f90 contains miscellaneous subroutines which do not fit well to other modules. The interface to the main PUMA module puma.f90 is given by the subroutines *miscini*, *miscstep* and *miscstop* which are called in puma.f90 from the subroutines *prolog*, *gridpointd* and *epilog*, respectively. A subroutine to eliminate spurious negative humidity and an optional subroutine to relax the upper level temperature towards a prescribed distribution is included in miscmod.f90.

Input/Output miscmod.f90 does not use any extra output file. If the relaxation is switched on, a climatological annual cycle of the prescribed upper level temperature distribution [K] is read from the external file **CLIMATEFILE** (see namelist). The file format is formatted SERVICE format with (8I10) for the headers and (8E12.6) for the temperature fields. To assign the field, the header needs to have the header information code 130, level 1 and a date identifier of the form *yymmdd* or *mmdd* where *mm* goes from 01 to 12 (January to December). Fields which are not needed will be skipped. The module is controlled by the namelist *miscpar* which is part of the namelist file **puma_namelist**:

Parameter	Type	Purpose	default
NFIXER	Integer	Switch for correction of negative moisture (0 = off , 1= on)	1
NUDGE	Integer	Switch for temperature relaxation in the uppermost level (0 = off , 1= on)	0
TNUDGE	Real	Time scale [d] of the temperature relaxation	10.
CLIMATEFILE	Character	Name of the file containing the prescribed temperature distribution	surface_parameter

Structure Internally, miscmod.f90 uses the FORTRAN-90 module *miscmod*, which uses the global common module *pumamod* from pumamod.f90. Subroutine *miscini* reads the namelist and, if the parallel version is used, distributes the namelist parameters to the different processes. If the relaxation is switched on, the climatological temperature is read from **CLIMATEFILE** and distributed to the processors. Subroutine *miscstep* calls the subroutine *fixer* to eliminate spurious negative humidity arising from the spectral method and, if the relaxation is switched on, calls the subroutine *mknudge* to do the temperature nudging. Subroutine *miscstop* is a dummy subroutine since there is nothing to do to finalize the computations in miscmod.f90.

2.3 surfmod.f90

General The module surfmod.f90 deals as an interface between the atmospheric part of the model and modules, or models, for the land and the oceans. The interface to the main PUMA module puma.f90 is given by the subroutines *surfini*, *surfstep* and *surfstop* which are called in puma.f90 from the subroutines *prolog*, *gridpointd* and *epilog*, respectively. Calls to subroutines named *landini*, *landstep* and *landstop* and *seaini*, *seastep* and *seastop* provide the interface to land and the ocean modules, respectively.

Input/Output surfmod.f90 reads the land-sea mask [frac.] and the orography (surface geopotential) [m^2/s^2] from file **INPUTFILE** (see namelist). The file format is formatted SERVICE format with (8I10) for the headers and (8E12.6) for the fields. To assign the fields, the headers need to have the header information code 129 for the surface geopotential and 172 for the land-sea mask (1. = land; 0. = sea). Fields which are not needed will be skipped. surfmod.f90 is controlled by the namelist *surfpar* which is part of the namelist file **puma_namelist**:

Parameter	Type	Purpose	default
NSURF	Integer	Debug switch	not active
NOROMAX	Integer	Resolution of orography	NTRU
OROSCALE	Real	Scaling factor for orography	1.0
INPUTFILE	Character	Name of the input file	surface_parameter

Structure Internally, surfmod.f90 uses the FORTRAN-90 module *surfmod*, which uses the global common module *pumamod* from pumamod.f90. Subroutine *surfini* reads the namelist and, if the parallel version is used, distributes the namelist parameters to the different processes. If the run is not started from a restart file (NRESTART from namelist *inp*of puma.f90 is 0), the land-sea-mask and the orography are read from file **INPUTFILE**. According to the namelist input, the orography is scaled by OROSCALE, transfered into spectral space and truncated to NOROMAX. Calls to subroutines *landini* and *seaini* are the interfaces to the respective initialization routines contained in the land and ocean modules. During the run, the interface to land and ocean is given by calls to the external subroutines *landstep* and *seastep*, which are called by *surfstep*. At the end of the integration, interface subroutines *landstop* and *seastop* are called by *surfstop*.

2.4 fftmod.f90

General The module fftmod.f90 contains all subroutines necessary to perform the fast fourier transformation and its inverse. The interface to the main PUMA module puma.f90 and to other modules (at the moment surfmod.f90, legmod.f90 and rainmod.f90) is given by the subroutines *gp2fc* and *fc2gp* which are called in puma.f90 from the subroutines *gridpointa* and *gridpointd*, in surfmod.f90 from *surfini*, in legmod.f90 from *sp2gp*, and in rainmod.f90 from *mkdqdtgp*.

Input/Output fftmod.f90 does not use any extra input file or output file. No namelist input is required.

Structure Internally, fftmod.f90 uses the FORTRAN-90 module *fftmod*, which uses no other modules. Subroutine *gp2fc* performs the transformation from grid point space into fourier space while the subroutine *fc2gp* does the transformation from fourier space into grid point space. Both routines use several subroutines to do the direct or indirect transformation for different factors. When *gp2fc* or *fc2gp* is called the first time, *fftini* is called to do the initialization of the FFT.

2.5 landmod.f90

General The module landmod.f90 contains parameterizations for land surface and soil processes which include the simple biome model SIMBA and a model for the river runoff. The interface to PUMA is given via the module surfmod.f90 by the subroutines *landini*, *landstep* and *landstop* which are called in surfmod.f90 from the subroutines *surfini*, *surfstep* and *surfstop*, respectively.

Input/Output landmod.f90 reads several surface and soil parameters either from the initial file **STARTFILE** (see namelist) or from the restart file **land_restart** which is written at the end of an integration. **STARTFILE** contains different surface fields which are needed for initialization. The file format is formatted SERVICE format with (8I10) for the header and (8E12.6) for the fields. The file may include the following fields: surface geopotential (orography) [m^2/s^2], land-sea mask [frac.], surface roughness [m], background albedo [frac.], glacier mask [frac.], bucket size [m], soil temperature [K], climatological annual cycle of the surface temperature [K], climatological annual cycle of the soil wetness [m]. To assign the fields, the headers need to have the header information code 129 for surface geopotential, code 172 for the land-sea mask (1. = land; 0. = sea), 173 for the surface roughness, 174 for the background albedo, 232 for the glacier mask (1. = glacier; 0. = no glacier), 229 for the bucket size, 209 for the soil temperature, 169 for the surface temperature and 140 for the soil wetness. for the climatological annual cycles of surface temperature and soil wetness, a date identifier of the form *yymmdd* or *mmdd* where *mm* goes from 01 to 12 (January to December) is required. Fields which are not needed will be skipped. If there are some fields not present in the **STARTFILE** default values will be used which can be set in the namelist. The use of some fields depend on the setting of some namelist parameters. The restart file **land_restart** is an unformatted file which contains all variables needed to continue the run. landmod.f90 is controlled by the namelist *landpar* given in the namelist file **land_namelist**:

Parameter	Type	Purpose	Default
NLANDT	Integer	Switch for surface temperature (1 = computed; 2 = climatology)	1
NLANDW	Integer	Switch for soil wetness (1 = computed; 2 = climatology)	1
NBIOME	Integer	Switch for biome model SIMBA (1 = on ; 0 = off)	0
ALBLAND	Real	Background albedo	0.2
DZ0LAND	Real	Roughnesslength [m]	2.0
DRHSLAND	Real	Wetness factor	0.25
ALBSMIN	Real	Minimum albedo for snow	0.4
ALBSMAX	Real	Maximum albedo for snow	0.8

Parameter	Type	Purpose	Default
ALBGMIN	Real	Minimum albedo for glaciers	0.6
ALBGMAX	Real	Maximum albedo for glaciers	0.8
WSMAX	Real	Maximum field capacity of soil water (bucket size) [m]	0.5
DRHSFULL	Real	Threshold above which wetness factor is 1	0.4
DZGLAC	Real	Threshold of orography to be glacier (-1.0 = none) [m]	-1.0
DZTOP	Real	Thickness of the uppermost soil layer [m]	0.2
DSOILZ(5)	Real Array	Soil layer thicknesses [m]	0.4,0.8,1.6,3.2,6.4
STARTFILE	Character	Initialization file	surface_parameter

Structure Internally, landmod.f90 uses the FORTRAN-90 module *landmod*, which uses the global common module *pumamod* from pumamod.f90. Subroutine *landini* reads the namelist and, if the parallel version is used, distributes the namelist parameters to the different processes. If the run is not started from a restart file (NRESTART from namelist *inp* of puma.f90is 0), the initialization file **STARTFILE** is being read. The soil and the river runoff are initialized via *soilini* and *roffini* and different variables are set according to the values given by the namelist or the **STARTFILE**. If it is a restart (NRESTART = 1), the restart records are being read from **land_restart**. Additionally, the climatological surface temperatures and soil wetnesses are updated from **STARTFILE** if NRESTART = 2. If NRESTART = 3 (special application) the bucket size, the roughness length and the albedo are set to the values given in the namelist. Subroutine *landstep* computes new surface and soil values via *soilstep* which calls *tands* and *wandr* for the heat and water budgets, respectively. If NLANDT and/or NLANDW are set to 0, climatological values are used for the surface temperature and the soil wetness. Via *roffstep* the river runoff is computed. Finally the biome model *simbastep* is called. The land model is finalized by *landstop* which writes the restart record to **land_restart**.

2.6 legmod.f90

General The module legmod.f90 contains all subroutines necessary to perform the Legendre transformation and its inverse. The interface to the main PUMA module puma.f90 and to other modules (at the moment surfmod.f90 and rainmod.f90) is given by the subroutines *legini*, *gauaw*, *sp2fl*, *invlega*, *invlegd*, *fc2sp*, *dirlega*, *dirlegd*, *fc3sp*, *uv2dv* and *sp2gp* which are called in puma.f90 from the subroutines *prolog*, *gridpointa* and *gridpointd*, in surfmod.f90 from *surfini*, and in rainmod.f90 from *mkdqdtgp*.

Input/Output legmod.f90 does not use any extra input file or output file. No namelist input is required

Structure Internally, legmod.f90 uses the FORTRAN-90 module *legmod*, which uses the global common module *pumamod* from pumamod.f90. Subroutine *legini* does the initialization. Subroutine *gauaw* computes the Gaussian latitudes and the corresponding weights. Subroutine *sp2fl* performs the transformation from spectral to fourier space for multilevel fields. The Subroutines *invlega* and *invlegd* contain the transformations from spectral to fourier space for all varibles which are needed in the adiabatic and in the diabatic part of the model, respectively. Subroutine *fc2sp* does the transformation from fourier to spectral space. The Subroutines *dirlega* and *dirlegd* contain the transformations from fourier to spectral space for the tendencies computed in the adiabatic and in the diabatic part of the model, respectively. Subroutine *uv2dv* transforms the fourier coefficients of the zonal and meridional wind components to spectral coefficients of divergence and vorticity.

2.7 mpimod.f90 and mpimod_dummy.f90

General The module mpimod.f90 contains interface subroutines to the MPI (Message Passing Interface) needed for massive parallel computing. Several MPI routines are called from the module. The interface to other modules are given by numerous subroutines which name starts with *mp*. Subroutines from mpimod.f90 are called in fluxmod.f90, icemod.f90, landmod.f90, miscmod.f90, oceanmod.f90, oceanmod50.f90, outmod.f90, puma.f90, radmod.f90, rainmod.f90, seamod.f90, surfmod.f90 and visumod.f90. The module mpimod_dummy.f90 is used instead of mpimod.f90 for simulations on a single processor. mpimod_dummy.f90 contains subroutines having the same name as the corresponding routine in mpimod.f90. However, there is no interface to MPI present in these routines and most of them are dummies.

Input/Output mpimod.f90 and mpimod_dummy do not use any extra input file or output file. No namelist input is required

Structure Internally, mpimod.f90 uses the FORTRAN-90 module *mpimod*, which uses the global common module *pumamod* from pumamod.f90 and the MPI module *mpi*. mpimod_dummy.f90 does not use any module. The following subroutines are included in mpimod.f90:

Subroutine	Purpose
mpbci	broadcast 1 integer
mpbcin	broadcast n integers
mpbcr	broadcast 1 real
mpbcrn	broadcast n reals
mpscin	scatter n integers
mpscrn	scatter n reals
mpscgp	scatter grid point field
mpgagp	gather grid point field
mpgallgp	gather grid point field to all
mpscsp	scatter spectral field
mpgasp	gather spectral field
mpgacs	gather cross section
mpgallsp	gather spectral field to all
mpsum	sum spectral field
mpsumsc	sum and scatter spectral field
mpsumr	sum n reals
mpsumbcr	sum and broadcast n reals
mpstart	initialize MPI
mpstop	finalize MPI

Subroutine	Purpose
mpreadgp	read and scatter grid point field
mpwritegp	gather and write grid point field
mpwritegph	gather and write (with header) grid point field
mpreadsp	read and scatter spectral field
mpwritesp	gather and write spectral field
mpi_info	give information about setup

2.8 outmod.f90

General The module outmod.f90 controls the data output of the model. The interface to the main PUMA module puma.f90 is given by the subroutines *outini*, *outgp*, *outsp*, *outreset* and *outaccu* which are called in puma.f90 from the subroutines *prolog* and *master*.

Input/Output outmod.f90 writes the output data to the file **puma_output** which is an unformatted file. **puma_output** is designed to be post processed by the AFTERBURNER program (see EDI) which converts the model variables to useful output in user friendly format. There is no separate namelist for outmod.f90 but some parameter of namelist *inp* of puma.f90 are used to control the format and the output interval.

Structure Internally, outmod.f90 uses the global common module *pumamod* from pumamod.f90 in several subroutines. Subroutine *outini* does the initialization. Subroutines *outgp* and *outsp* write the grid point and the spectral fields to the output file **puma_output**. *outaccu* accumulates some output variables over the output interval. *outreset* resets the accumulated arrays to zero.

2.9 puma.f90

General The module puma.f90 is the main module of the model. It includes the main program *puma* and controls the run. From puma.f90 the interface routines to the modules miscmod.f90, fluxmod.f90, radmod.f90, rainmod.f90, surfmod.f90 are called. The output is done by calling the interface routines to outmod.f90. In addition, the adiabatic tendencies and the horizontal diffusion are computed in puma.f90. To do the necessary transformations, calls to the modules fftmod.f90 and legmod.f90 are used.

Input/Output puma.f90 does not use any extra input file or output file. A diagnostic print out is written on standard output. puma.f90 is controlled by the namelist *inp* which is part of the namelist file **puma_namelist**:

Parameter	Type	Purpose	Default
KICK	Integer	Switch for initial white noise disturbance on surface pressure (0 = none; 1 = global; 2 = hemispherically symmetric; 3 = one wavenumber only)	1
NAFTER	Integer	Time interval for output [time steps]	12
NCOEFF	Integer	Number of spectral coefficients in diagnostic print out	0
NDEL(NLEV)	Integer Array	Order of the horizontal diffusion	NLEV · 2
NDIAG	Integer	Time interval for diagnostic print out [time steps]	12
NEXP	Integer	Experiment identifier	0
NEXPER	Integer	Switch for predefined experiments (not used)	0
NKITS	Integer	Number of initial explicit Euler time steps	3
NRESTART	Integer	Switch for restart (0 = initial run; 1 = normal restart; 2 = restart plus update of surface climatology; 3 = restart plus update of surface parameter (see landmod.f90))	0
NRUN	Integer	Number of time steps to be run	0
NSTEP	Integer	Current time step (replaced by restart record)	0

Parameter	Type	Purpose	Default
NSTOP	Integer	Finishing time step	0 (= not active)
NTSPD	Integer	Number of time steps per day	24
NDAYS	Integer	Number of days to be run (overwrites NRUN if > 0)	0
NEQSIG	Integer	Switch for non equally spaced sigma levels (1 = non equally spaced; 1 = equally spaced)	1
NPRINT	Integer	Switch for extended debug print out (0 = off; 1 = on; 2 = very extended)	0
NPRHOR	Integer	Number of the grid point to be used for very extended debug print out	0
NPACKSP	Integer	Switch for spectral output (0 = normal; 1 = compressed)	1
NPACKGP	Integer	Switch for grid point output (0 = normal; 1 = compressed)	1
NRAD	Integer	Switch for radiation (0 = off; 1 = on)	1
NFLUX	Integer	Switch for surface fluxes and vertical diffuson (0 = off; 1 = on)	1
NDIAGGP	Integer	Switch for additional diagnostic grid point output (0 = off; 1 = on)	0
NDIAGSP	Integer	Switch for additional diagnostic spectral output (0 = off; 1 = on)	0
NDIAGCF	Integer	Switch for additional cloud forcing diagnostic (0 = off; 1 = on)	0
NDIAGGP2D	Integer	Number of additional diagnostic 2-d grid point output (0 = off; 1 = on)	0
NDIAGGP3D	Integer	Number of additional diagnostic 3-d grid point output (0 = off; 1 = on)	0
NDIAGSP2D	Integer	Number of additional diagnostic 2-d spectral output (0 = off; 1 = on)	0
NDIAGSP3D	Integer	Number of additional diagnostic 3-d spectral output (0 = off; 1 = on)	0

Parameter	Type	Purpose	Default
NDL(NLEV)	Integer Array	Switch for diagnostic print out of a level (0 = off; 1 = on)	NLEV · 0
NHDIFF	Integer	Cut off wave number for horizontal diffusion	15
NTIME	Integer	Switch for CPU time diagnostics (0 = off; 1 = on)	0
NPERPETUAL	Integer	Switch for perpetual simulations (0 = annual cycle; >0 = day of the year)	0
DTEP	Real	Equator to pole temperature difference [K] for Newtonian cooling (usually not used)	0.0
DTNS	Real	North pole to south pole temperature difference [K] for Newtonian cooling (usually not used)	0.0
DTROP	Real	Tropopause height [m] for Newtonian cooling (usually not used)	12000.0
DTTRP	Real	Smoothing of the tropopause [K] for Newtonian cooling (usually not used)	2
TGR	Real	Surface temperature [K] for Newtonian cooling (usually not used)	280
TDISSD(NLEV)	Real Array	time scale [d] for the horizontal diffusion of divergence	NLEV · 0.2
TDISSZ(NLEV)	Real Array	time scale [d] for the horizontal diffusion of vorticity	NLEV · 1.1
TDISST(NLEV)	Real Array	time scale [d] for the horizontal diffusion of temperature	NLEV · 5.6
TDISSQ(NLEV)	Real Array	time scale [d] for the horizontal diffusion of moisture	NLEV · 5.6
PSURF	Real	Global averaged sea level pressure [Pa]	101325.00
RESTIM(NLEV)	Real Array	Time scale [d] for Newtonian cooling (usually not used)	NLEV · 0.0

Parameter	Type	Purpose	Default
T0(NLEV)	Real Array	Reference temperature used in the discretization scheme	NLEV · 250.0
TFRC(NLEV)	Real Array	Time scale [d] for Rayleigh friction (0.0 = off)	NLEV · 0.0

Structure Internally, fluxmod.f90 uses the FORTRAN-90 global common module *pumamod* from pumamod.f90. After starting MPI, the main program *puma* calls *prolog* for initializing the model. Then, *master* is called to do the time stepping. Finally, subroutine *epilog* finishes the run. In subroutine *prolog*, calls to different subroutines, which are part of puma.f90 or are provided by other modules, initialize various parts of the model: *gauaw* and *inilat* build the grid, *readnl* reads the namelist and sets some parameter according to the namelist input, *initpm* and *initsi* initialize some parameter for the physics and the semi implicit scheme, respectively. *outini* starts the output. If NRESTART > 0, the restart record is read by *restart*, otherwise *initfd* sets the prognostic variables to their initial values. calls to *miscini* *fluxini*, *radini*, *rainini* and *surfini* start the initialization of the respective external modules. Finally, the global averaged surface pressure is set according to PSURF and the orography. Subroutine *master* controls the time stepping. First, if its not a restart, initial NKITS explicit forward timesteps are performed. The main loop is defined by calling *gridpointa* for the adiabatic tendencies, *spectrala* to add the adiabatic tendencies, *gridpointd* for the diabatic tendencies (which are computed by the external modules), *spectrald* to add the diabatic tendencies and the interface routines to the output module outmod.f90. The run is finalized by subroutine *epilog* which writes the restart records and calls the respective interface routines of the external modules.

2.10 pumamod.f90

General The module pumamod.f90 contains all parameters and variables which may be used to share information between puma.f90 and other modules. No subroutines or programs are included.

Input/Output pumamod.f90 does not use any extra input file or output file. No namelist input is required

Structure Internally, pumamod.f90 is a FORTRAN-90 module named *pumamod*. Names for global parameters, scalars and arrays are declared and, if possible, values are preset.

2.11 radmod.f90

General The module radmod.f90 contains subroutines to compute radiative energy fluxes and the temperature tendencies due to long wave and short wave radiation. The interface to the main PUMA module puma.f90 is given by the subroutines *radini*, *radstep* and *radstop* which are called in puma.f90 from the subroutines *prolog*, *gridpointd* and *epilog*, respectively.

Input/Output radmod.f90 does not use an extra output file. If the Switch for ozone (NO3, see namelist) is set to 2 (externally prescribed), the climatological cycle of the ozone distribution is read from the external file **OZONEFILE** which name is given in the namelist. The file format is formatted SERVICE format with (8I10) for the header and (8E12.6) for the fields. To assign the fields, the headers need to have the header information code 200, level going from 1 to NLEV and a date identifier of the form *yymmdd* or *mmdd* where *mm* goes from 01 to 12 (January to December). radmod.f90 is controlled by the namelist *radpar* which is part of the namelist file **puma_namelist**:

Parameter	Type	Purpose	Default
NDCYCLE	Integer	Switch for diurnal cycle of insolation (0 = off, 1 = on)	0
NO3	Integer	Switch for ozone (0 = off, 1 = idealized distribution, 2 = externally presrcibed)	1
CO2	Real	CO_2 concentration [ppmv]	360.0
GSOL0	Real	Solar constant [W/m^2]	1367.0
IYRBP	Integer	Year PB (reference is 1950) to calculate orbit from	-50
NSWR	Integer	Switch for short wave radiation (0 = off, 1 = on)	1
NLWR	Integer	Switch for long wave radiation (0 = off, 1 = on)	1
NSOL	Integer	Switch for incoming solar radiation (0 = off, 1 = on)	1
NSWRCL	Integer	Switch for computed short wave cloud properties (0 = off, 1 = on)	1
NRSCAT	Integer	Switch for Rayleigh scattering (0 = off, 1 = on)	1
RCL1(3)	Real Array	Prescribed cloud albedos [frac.] for high, middle and low level clouds (spectral range 1)	0.15,0.30.0.60

Parameter	Type	Purpose	Default
RCL2(3)	Real Array	Prescribed cloud albedos [frac.] for high, middle and low level clouds (spectral range 2)	0.15,0.30.0.60
ACL2(3)	Real Array	Prescribed cloud absorptivities [frac.] for high, middle and low level clouds (spectral range 2)	0.05,0.10.0.20
CLGRAY	Real	Prescribed grayness of clouds (-1.0 = computed)	-1.0
TPOFMT	Real	Tuning for point of mean transmission	0.15
ACLLWR	Real	Mass absorption coefficient for clouds (long wave)	0.1
TSWR1	Real	Tuning of cloud albedo (spectral range 1)	0.035
TSWR2	Real	Tuning of cloud back scattering (spectral range 2)	0.04
TSWR3	Real	Tuning of cloud single scattering albedo (spectral range 2)	0.006
OZONEFILE	Character	File for externally prescribed ozone distribution	ozone.dat

Structure Internally, radmod.f90 uses the FORTRAN-90 module *radmod*, which uses the global common module *pumamod* from pumamod.f90. Additionally, the FORTRAN-90 module *orbparam* is used. Subroutine *radini* reads the namelist and, if the parallel version is used, distributes the namelist parameters to the different processes. Orbital parameters are computed by calling *orb_params*. If NO3 is set to 2, the ozone distribution is read from **OZONEFILE**. Subroutine *radstep* calls the subroutines *solang* and *mko3* to compute the cosine of the solar angle and the ozone distribution, respectively. The short wave radiative fluxes are calculate in *swr* while the long wave radiative fluxes are computed in *lwr*. Subroutine *radstop* is a dummy subroutine since there is nothing to do to finalize the computations in radmod.f90.

2.12 rainmod.f90

General The module rainmod.f90 contains subroutines to compute large scale and convective precipitation and the related temperature tendencies. In addition, a parameterization of dry convective mixing of temperature and moisture is included and cloud cover is diagnosed. The interface to the main PUMA module puma.f90 is given by the subroutines *rainini*, *rainstep* and *rainstop* which are called in puma.f90 from the subroutines *prolog*, *gridpointd* and *epilog*, respectively.

Input/Output rainmod.f90 does not use any extra input or output file and is controlled by the namelist *rainpar* which is part of the namelist file **puma_namelist**:

Parameter	Type	Purpose	Default
KBETTA	Integer	Switch for betta in Kuo parameterization (0 = off, 1 = on)	1
NPRL	Integer	Switch for large scale precipitation (0 = off, 1 = on)	1
NPRC	Integer	Switch for convective precipitation (0 = off, 1 = on)	1
NDCA	Integer	Switch for dry convective adjustment (0 = off, 1 = on)	1
RCRIT(NLEV)	Real Array	Critical relative humidity for cloud formation	computed
CLWCRIT1	Real	Critical vertical velocity for cloud formation [Pa/s] (not active if CLWCRIT2 > CLWCRIT1)	-0.1
CLWCRIT2	Real	Critical vertical velocity for cloud formation [Pa/s] (not active if CLWCRIT2 > CLWCRIT1)	0.0

Structure Internally, rainmod.f90 uses the FORTRAN-90 module *rainmod*, which uses the global common module *pumamod* from pumamod.f90. Subroutine *rainini* reads the namelist and, if the parallel version is used, distributes the namelist parameters to the different processes. Subroutine *rainstep* calls the subroutine *mkdqdtgp* to obtain the adiabatic moisture tendencies in grid point space, which are needed for the Kuo parameterization. *kuo* is called to compute the convective precipitation and the respective tendencies. Dry convective adjustment is performed in *mkdca*. Large scale precipitation is computed in *mklsp*. Finally, diagnostic clouds are calculated in *mkclouds*. Subroutine *radstop* is a dummy subroutine since there is nothing to do to finalize the computations in radmod.f90.

2.13 seamod.f90

General The module seamod.f90 is the interface from the atmosphere to the ocean and the sea ice. The interface to the main PUMA module puma.f90 is given by the subroutines *seaini*, *seastep* and *seastop* which are called in puma.f90 from the subroutines *prolog*, *gridpointd* and *epilog* respectively.

Input/Output seamod.f90 reads different surface parameters either from the file **SSTFILE** (see namelist) and the file **ocean_parameter** or from the restart file **sea_restart** which is written at the end of an integration.. The files formats are unformatted for the restart file, formatted SERVICE format with (8I10) for the header and (8E12.6) for the fields for **SSTFILE** and formatted EXTRA format with (4I10) for the header and (6(1X,E12.6)) for the fields for **ocean_parameter**. The file **SSTFILE** may include the following fields: The climatological annual cycle of the surface temperature [K] and the climatological annual cycle of the sea ice compactness [frac.]. To assign the fields, the headers need to have the header information code 169 for surface temperature and code 210 for the compactness (1 = ice; 0. = open water). a date identifier of the form *yymmdd* or *mmdd* where *mm* goes from 1 to 12 (January to December) is required. Fields which are not needed will be skipped. The file **ocean_parameter** includes the following fields: The climatological annual cycle of the sea surface temperature [K], the climatological annual cycle of the mixed layer depth [m] and the climatological average of the deep ocean temperature [m]. To assign the fields, the order must be as described above (no header information is used). The restart file **sea_restart** contains all variables needed to continue the run. seamod.f90 is controlled by the namelist *seapar* given in the namelist file **sea_namelist**:

Parameter	Type	Purpose	Default
ALBSEA	Real	Albedo for ice free ocean	0.069
ALBICE	Real	Maximum albedo for sea ice	0.7
DZ0SEA	Real	Minimum roughness length [m] for ice free ocean	$1.0 \cdot 10^{-5}$
DZ0ICE	Real	Roughness length [m] for sea ice	$1.0 \cdot 10^{-3}$
DRHSSEA	Real	Wetness factor for ice free ocean	1.0
DRHSICE	Real	Wetness factor for sea ice	1.0
NOCEAN	Integer	Switch for ocean model (0 = climatological SST, 1 = ocean model)	1
NICE	Integer	Switch for sea ice model (0 = climatological, 1 = sea ice model)	1

Parameter	Type	Purpose	Default
NCPL_ICE_OCEAN	Integer	ice-ocean coupling time steps	32
NCPL_ATMOS_ICE	Integer	ice atmosphere coupling time steps	1
TDEEPSEA	Real	Homogeneous deep ocean temperature [K]	0.0
DHICEMIN	Real	Minimum sea ice thickness [m]	0.1
SSTFILE	FILE	File with SST data	surface_parameter

Structure Internally, seamod.f90 uses the FORTRAN-90 module *seamod*, which uses the global common module *pumamod* from pumamod.f90. Subroutine *seaini* reads the namelist and, if the parallel version is used, distributes the namelist parameters to the different processes. The coupling interface routines in intermod_atm.f90 are initialized by calling *cplinit*. If it is not a restart (i.e. if NRESTART from *inp* of puma.f90 is 0), the files **SSTFILE** and **ocean_parameter** are being read. The climatological sea ice compactness is converted to a sea ice thickness as initial condition and additional surface parameters are set. If it is a restart, the restart file **sea_restart** is read. Subroutine *seastep* accumulates the variables used for the coupling between the atmosphere and the ocean. The coupling is done via the sea ice model. There is no direct connection between atmosphere and ocean model. If there is no sea ice, the coupling quantities are passed through the ice model without changes. A call to *cplexchange_ice* from module intermod_atm.f90 transfers the atmospheric coupling fluxes to the sea ice model and gets the sea ice and ocean surface data back. After the call, additional surface parameter are computed. Subroutine *seastop* finalizes the run and writes the restart records.

2.14 Files

2.15 Sea ice and ocean modules

This section describes the modules that represent sea ice and ocean and the necessary interfaces between these modules and the atmospheric modules. Conceptually, the sea ice model lies inbetween the atmosphere model and the ocean model. Thus, the PUMA main part and the ocean model are both coupled to the sea ice model, but not directly to each other. The sea ice model decides whether a given gridpoint is covered with ice or not, in the latter case, it merely functions as passing the ocean fluxes to the atmosphere and vice versa. The parameters that are exchanged are listed in Table 2.1. The sea ice and ocean model use a time step of one day. Thus, atmospheric coupling to the sea ice model is performed every 32 time steps, while the sea ice and ocean model are coupled every time step. The coupling scheme is shown in Fig. 2.1. Fig. 2.2 shows how the subroutines are placed when no external coupler is used.

Parameter	Atmosphere $\leftarrow\rightarrow$ Ice	Ice $\leftarrow\rightarrow$ Ocean
Ice cover	$\leftarrow$	–
Ice thickness	$\leftarrow$	$\rightarrow$
Snow thickness	$\leftarrow$	$\rightarrow$
Surface temperature	$\leftarrow$	$\leftarrow$
Deep sea temperature	–	$\leftarrow$
Mixed layer depth	–	$\leftarrow$
Net precipitation, runoff	$\rightarrow$	$\rightarrow$
Salinity	–	$\leftarrow$
Melt and freeze volume	–	$\rightarrow$
Heat fluxes	$\rightarrow$	$\rightarrow$
d(Heat fluxes)/dT	$\rightarrow$	–
Radiation	$\rightarrow$	–
Wind stress	$\rightarrow$	$\rightarrow$

Table 2.1: Parameters to be exchanged between models. Arrows denote the direction in which the parameter is passed, e.g. the atmosphere receives ice cover information from the ice model.

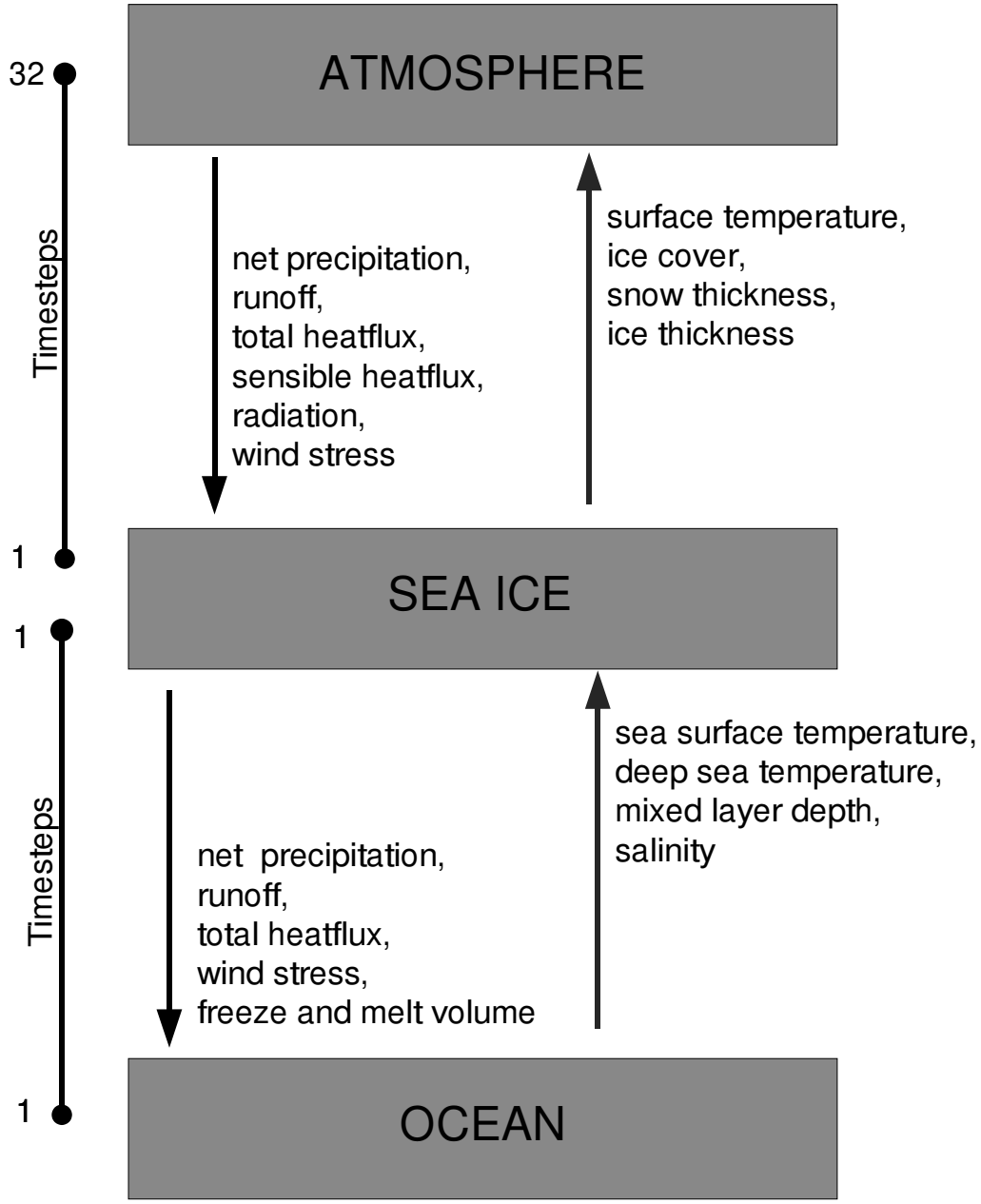


Figure 2.1: Schematic illustration of the model coupling.

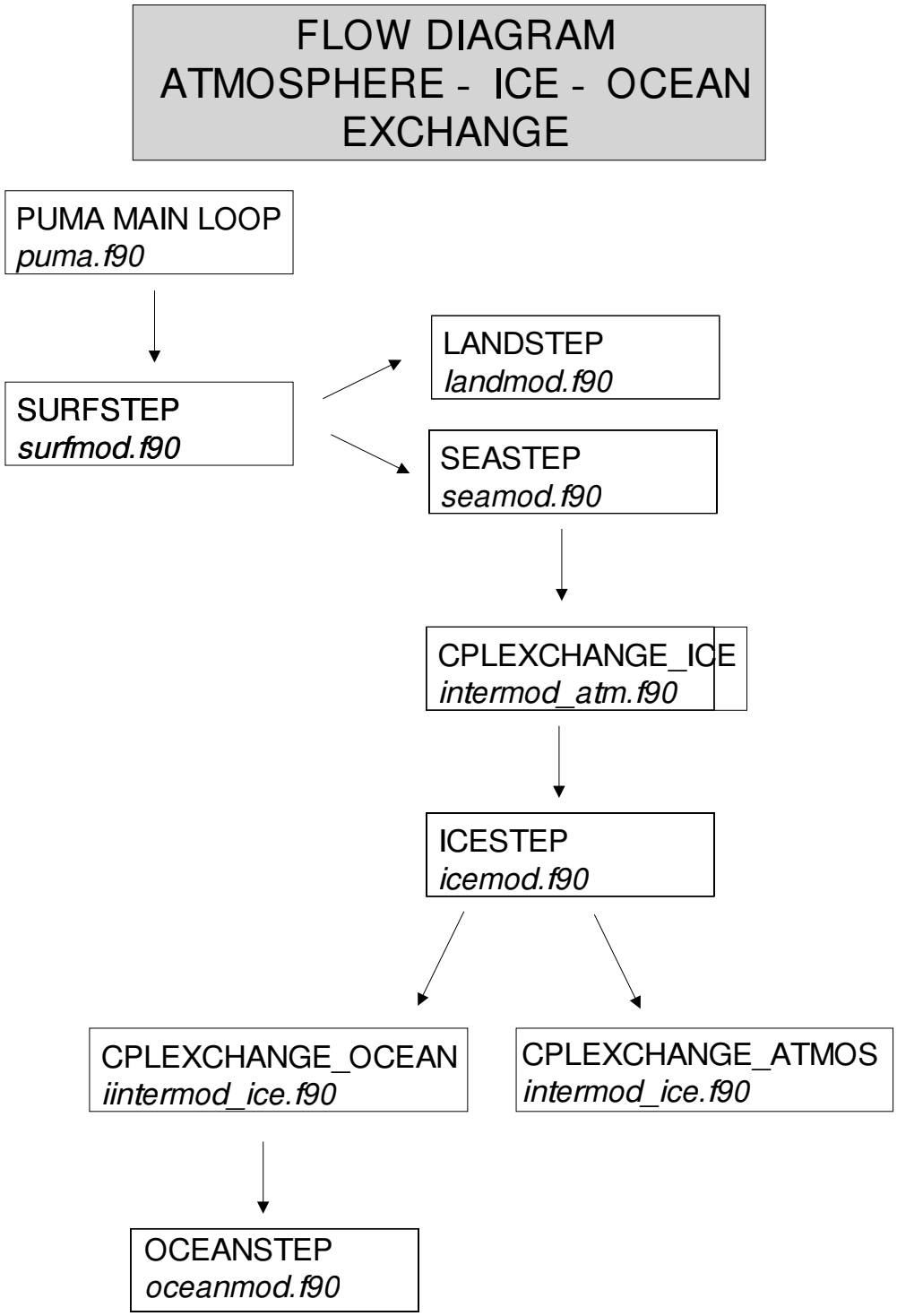


Figure 2.2: Subroutine flow when no external coupler is used.

2.15.1 seamod.f90

General The module seamod.f90 deals as an interface between the atmospheric part of the model and modules for the ocean and sea ice. The basic subroutines *seaini*, *seastep* and *seastop* are called by the the subroutines *surfini*, *surfstep* and *surfstop*, respectively (see module surfmod.f90). See the reference guide, section (heiko.coupling) for a visualization of the module coupling structure.

Input/Output seamod.f90 needs the parameter file **surface_parameter** to read the climatological sea surface temperature and ice cover as well as **ocean_parameter**, which contains climatological mixed layer depth and the Levitus 400 m temperature. As output data, the file **sea_restart** is produced at the end of a run. In the case of a restart, this file is required to be read in by the module. The namelist *seapar*, which is contained in the file **sea_namelist**, is defined as:

Parameter	Type	Purpose	default
ALBSEA	REAL	Albedo for open water	0.069
ALBICE	REAL	Max. albedo for sea ice	0.7
DZ0SEA	REAL	Roughness length sea	$1.5 \cdot 10^{-5} m$
DZ0ICE	REAL	Roughness length ice	$1.0 \cdot 10^{-3} m$
DRHSSEA	REAL	Wetness factor sea	1.0
DRHSICE	REAL	Wetness factor ice	1.0
NOCEAN	INTEGER	Ocean model (1) or climatology (0)	1
NICE	INTEGER	Sea ice model (1) or climatology (0)	1
NCPL_ICE_OCEAN	INTEGER	Ice Ocean coupling time steps	1
NCPL_ATMOS_ICE	INTEGER	Atmosphere Ice coupling time steps	32
SSTFILE	CHAR*80	file containing climatology "surface_parameter"	

Structure Internally, seamod.f90 uses the FORTRAN-90 module *seamod*, which uses the global common module *pumamod* from pumamod.f90. Subroutine *seaini* reads the namelist and, if the parallel version is used, distributes the namelist parameters to the different processes. If the run is not started from a restart file (NRESTART from namelist *inp* is 0), the sea surface temperature and the ice cover is read from the **surface_parameter** file. Ice thickness is computed from ice cover. Additionally, mixed layer depth and the 400 m Levitus temperature is read from the file **ocean_parameter**. Climatology and namelist information is passed to the ice and ocean modules via the external subroutines *iceini* (in icemod.f90) and *oceanini* (in oceanmod.f90 or oceanmod50.f90). Every NCPL_ATMOS_ICE time steps, *seastep* calls the ice module via the external subroutine *cplexchange_ice* (defined in intermod_atm.f90). At the end of the integration, *seastop* writes the restart information into file **sea_restart**.

2.15.2 intermodatm.f90

General The module intermod_atm.f90 contains subroutines that exchange information between the atmospheric module and the sea ice module. If an external coupler is used with an independent sea ice / ocean model, the module is replaced e.g. by mpccimod_atm.f90 which contains the relevant subroutines for the MpCCI coupler.

Input/Output intermod_atm.f90 does not use any extra input file or output file.

Structure The subroutines *cplstart*, *cplinit*, *cplstop* are dummy routines that are real subroutines only in the case of external coupling. The subroutine *cplexchange_ice*, which is called by *seastep* in module seamod.f90, calls the external subroutine *icestep* (defined in icemod.f90). It then copies the ice / ocean data to the relevant PUMA variables.

2.15.3 intermodice.f90

General The module intermod_ice.f90 contains subroutines that exchange information between the sea ice module and the ocean and atmosphere module. If an external coupler is used with an independent sea ice / ocean model, the module is replaced e.g. by mpccimod_ice.f90 which contains the relevant subroutines for the MpCCI coupler.

Input/Output intermod_ice.f90 does not use any extra input file or output file.

Structure The subroutine *cplexchange_ocean*, which is called by *icestep* in module icemod.f90, calls the external subroutine *oceanstep* (defined in oceanmod.f90) if the *sea_namelist* entry NOCEAN is set to 1. Otherwise, it calls the subroutine *oceanget* (defined in oceanmod.f90), which interpolates the climatological values to the current time step. It then returns the ocean data to the subroutine *icestep*. The subroutine *cplexchange_atmos*, which is also called by *icestep* in module icemod.f90, copies the atmospheric forcing data to the relevant variables defined in *icemod*.

2.15.4 icemod.f90

General The module icemod.f90 contains subroutines to compute sea ice cover and thickness. The interface to the main PUMA module is given by the subroutine *icestep*, which is called by *cplexchange_ice* (defined in intermod_atm.f90), which is called by *seastep* (defined in seamod.f90).

Input/Output icemod.f90 requires the file **ice_flxcor** if NFLXCORR is set to a negative value. If NOUTPUT is set to 1, the output files **fort.75** containing global fields of ice model data and the file **fort.76** containing diagnostic ice data are produced (for details, see the reference guide). Both output files are in service format. The module is controlled by the namelist *icepar* in the file **ice_namelist**.

Parameter	Type	Purpose	default
NDIAG	INTEGER	Diagnostic output every NDIAG time steps	160
NOUT	INTEGER	Model data output every NOUT time steps	32
NOUTPUT	INTEGER	Icemodel output (0=no,1=yes)	1
NFLXCORR	INTEGER	Time constant for restoring (> 0), no flux correction ($= 0$), use flux-correction from file (< 0)	$360\,d$

Structure icemod.f90 uses the module *icemod* which is not dependent on the module *pumamod*. Subroutine *iceini* reads the namelist and, when required, the flux correction from the file **ice_flxcor**. Subroutine *icestep* calls *cplexchange_atmos* (defined in intermod_ice) to get the atmospheric forcing fields. If the *sea_namelist* parameter NICE is set to 1, the subroutine *subice* is called, which calculates ice cover and thickness. Otherwise, climatological data, interpolated to the current time step by *iceget* are used. If an ice cover is present, the surface temperature is calculated in *skintemp*. Otherwise, the surface temperature is set to the sea surface temperature calculated by the ocean model. Every NCPL_ICE_OCEAN (defined in *sea_namelist*) time steps, the external subroutine *cplexchange_ocean* (defined in intermod_ice) is called to pass the atmospheric forcing to and retrieve oceanic data from the ocean module oceanmod.f90. The oceanic data is used for ice calculations in the next time step.

2.15.5 oceanmod.f90

General The module oceanmod.f90 contains a mixed layer ocean model, i.e. subroutines to compute sea surface temperature and mixed layer depth. The interface to the main PUMA module is via the module icemod.f90 given by the subroutine *oceanstep*, which is called by *cplexchange_ocean* (defined in intermod_ice).

Input/Output oceanmod.f90 requires the file **ocean_flxcor** if NFLXCORRSST or NFLXCORRMLD is set to a negative value. If NOUTPUT is set to 1, the output file **fort.31** containing global fields of ocean model data in service format is produced (for details, see the ice modul section of the reference guide). The module is controlled by the namelist *oceanpar* in the file **ocean_namelist**.

Parameter	Type	Purpose	default
NDIAG	INTEGER	Diagnostic output every NDIAG time steps	480
NOUT	INTEGER	Model data output every NOUT time steps	32
NOUTPUT	INTEGER	Oceanmodel output (0=no,1=yes)	1
NFLXCORRMLD	INTEGER	Time constant for restoring mixed layer depth (> 0), no flux correction ($= 0$), use fluxcorrection from file (< 0)	$60\,d$
NFLXCORRSST	INTEGER	Time constant for restoring sea surface temperature (> 0), no flux correction ($= 0$), use fluxcorrection from file (< 0)	$60\,d$

Structure oceanmod.f90 uses the module *oceanmod* which is not dependent on the module *pumamod*. Subroutine *oceanini* reads the namelist and, when required, the flux corrections from the file **ocean_flxcor**. Subroutine *oceanstep* calls *mixocean*, which calculates mixed layer depth and temperature. If an ice cover is present, mixed layer depth is set to the climatological value and the sea surface temperature is set to the freezing temperature. For details of the mixed layer model, see the reference guide section (ute).

2.15.6 oceanmod50.f90

General The module oceanmod50.f90 contains a mixed layer ocean model with depth fixed to 50 m and a SST fluxcorrection that does not stem from restoring to climatological data. For details, see section (ute) of the reference guide. The module oceanmod50.f90 optionally replaces the module oceanmod.f90, so the internal structure and the interface to the main PUMA module is identical to oceanmod.f90.

Input/Output If NFLXCORRSST is set to a negative value, oceanmod50.f90 requires the file **ocean_lgflxcor** (denoting long-term gradient flux correction). Otherwise, the file **heat_parameter** is needed to calculate the flux correction. If NOUTPUT is set to 1, the output file **fort.31** containing global fields of ocean model data in service format is produced (for details, see the ice module section of the reference guide). The module is controlled by the namelist *oceanpar* in the file **ocean_namelist**.

Parameter	Type	Purpose	default
NDIAG	INTEGER	Diagnostic output every NDIAG time steps	480
NOUT	INTEGER	Model data output every NOUT time steps	32
NOUTPUT	INTEGER	Oceanmodel output (0=no,1=yes)	1
NFLXCORRSST	INTEGER	Flag for calculating the flux correction (< 0) or reading the flux correction from file (> 0)	-1

Structure The internal structure is exactly the same as in oceanmod.f90.

Chapter 3

Running Planet Simulator

3.1 Interactive Console Mode

The Planet Simulator is started from a console by simply typing

```
puma.x
```

The following files have to be present in the same directory:

```
puma_namelist
land_namelist
sea_namelist
ice_namelist
ocean_namelist
surface_parameter
ocean_parameter
```

All settings, like length of the integration, special parameterizations etc. are given in the namelist files. The parameter files contain the climatology. When the integration is finished successfully, the following files have been created:

```
puma_output
puma_restart
land_restart
sea_restart
```

The file `puma_output` contains the model results and has to be postprocessed using the pumaburner (cf. Chapter **??**). The `_restart` files contain information necessary to restart the model run from the end of the current integration.

3.2 Batch Mode

For long integrations, it is more useful to run puma in batch mode, i.e. start puma by calling a script that manages the model run. The following script does just that. Since it is quite long, it is here split to parts with explanations inbetween.

```
#!/usr/bin/ksh
#
stime=‘date‘
#=========================
# This script runs the atmospheric model PUMA on a linux machine
#
EXP=example                              # EXPERIMENT IDENTIFIER
EXPDIR=/castor/home/user/${EXP}          # EXPERIMENT DIRECTORY
MODEL=${EXPDIR}/puma.x                   # THE MODEL EXECUTABLE
SCHAUER=1                                # TRANSFER OUTPUT TO SCHAUER (1=YES)
SCHAUERDIR=/pf/u/user_account/puma/${EXP} # U-TREE DIRECTORY (FOR OUTPUT)
DATADIR=${EXPDIR}/data                   # OUTPUT DIRECTORY
SSTFILE=${EXPDIR}/surface_parameter      # INITIAL DATAFILE (PUMA)
SURFFILE=${EXPDIR}/surface_parameter     # INITIAL DATAFILE (PUMA)
OCEANFILE=${EXPDIR}/ocean_parameter      # INITIAL DATAFILE (OCEANMOD)
LASTYEAR=50                              # LAST YEAR TO BE SIMULATED
FTPINT=12                                # MONTHS PER TAR-FILE
TMPDIR=${MFHOME}/tmpdir/run$$
#
mkdir -p ${TMPDIR}
cd $TMPDIR
set -ex
mkdir -p ${EXPDIR}
mkdir -p ${DATADIR}
#
OUTYEAR=1
OUTDAY=30
OUTMON=2
OUTFTP=2
TARFILE=${EXP}TAR_0101
#
cp $MODEL $EXPDIR/model.x
#
```

This first block of the script defines the basic settings, i.e. the directories used, the length of the integration etc. If `SCHAUER` is set to 1, all puma output is transferred to the specified directory on the schauer, thus avoiding the users directory from filling up. Otherwise, the output is written to the `DATADIR` directory. A temporary directory is created where the model is eventually run.

```
#
```

```
cat > $EXPDIR/NAMLIST.exe << 'EOX'
#
#  NAME LIST PARAMETER
#
cat > puma_namelist << EOF
 &INP
 NDAYS=30,
 NTSPD=32,
 NRESTART=${1},
 NDIAG=480,
 NAFTER=32,
 NEQSIG=1,
 PSURF=101325.,
 NPACKSP=0,
 NPACKGP=0,
 &END
 &MISCPAR
 &END
 &FLUXPAR
 &END
 &RADPAR
 &END
 &RAINPAR
 &END
 &SURFPAR
 &END
EOF
#
EOX
#
cat > ${EXPDIR}/land_namelist << EOL
 &landpar
 &end
EOL
cat > ${EXPDIR}/sea_namelist << EOO
 &seapar
 &end
EOO
cat > ${EXPDIR}/ocean_namelist << EOO
 &oceanpar
 &end
EOO
cat > ${EXPDIR}/ice_namelist << EOO
 &icepar
 &end
EOO
```

```
#
chmod u+x ${EXPDIR}/NAMLIST.exe
#
```

Now, the necessary namelists are generated. The puma namelist is defined as an executable which can be called with a parameter setting the restart mode.

```
$EXPDIR/NAMLIST.exe 0
cp ${EXPDIR}/land_namelist land_namelist
cp ${EXPDIR}/sea_namelist sea_namelist
cp ${EXPDIR}/ice_namelist ice_namelist
cp ${EXPDIR}/ocean_namelist ocean_namelist
#
cp $EXPDIR/model.x model.x
cp ${SSTFILE} surface_parameter
cp ${SURFFILE} surface_parameter
cp ${OCEANFILE} ocean_parameter
#
model.x > ${EXPDIR}/${EXP}PROUT_0101
#
mv puma_output ${EXP}PUMA_0101
#
# history and restart saved for further diagnostics
#
tar -cf ${EXPDIR}/${TARFILE} ${EXP}PUMA_0101
mv puma_restart $EXPDIR/${EXP}RES
mv land_restart $EXPDIR/${EXP}LANDRES
mv sea_restart $EXPDIR/${EXP}SEARES
#
echo ${OUTYEAR} > ${EXPDIR}/saveyear.${EXP}
echo ${OUTDAY} > ${EXPDIR}/saveday.${EXP}
echo ${OUTMON} > ${EXPDIR}/savemon.${EXP}
echo ${OUTFTP} > ${EXPDIR}/saveftp.${EXP}
echo ${TARFILE} > ${EXPDIR}/savetar.${EXP}
#
# cat runall to EXPDIR
#
cat > $EXPDIR/runall << EOR
#!/usr/bin/ksh
#
TMPDIR=${TMPDIR}
mkdir -p \${TMPDIR}
cd \$TMPDIR
set -ex
#
EXPDIR=$EXPDIR
SCHAUER=$SCHAUER
```

```
SCHAUERDIR=$SCHAUERDIR
DATADIR=$DATADIR
EXP=$EXP
LASTYEAR=$LASTYEAR
MONTHS=$MONTHS
FTPINT=$FTPINT
```

The puma namelist is generated with `NRESTART=0`, i.e. the first month is integrated from climatology. The script `runall` is generated, which can be used to restart the run after an interruption.
The remainder of the script is a loop of one-month integrations until the desired integration time is reached. After each year, the monthly output is tarred together and moved to the schauer or the data directory.

```
#
cp \${EXPDIR}/model.x model.x
#
#####################################################################
#
II=1
while [ \$II -le 2 ]
do
#
INMON=\‘cat \${EXPDIR}/savemon.\${EXP}\‘
INYEAR=\‘cat \${EXPDIR}/saveyear.\${EXP}\‘
INDAY=\‘cat \${EXPDIR}/saveday.\${EXP}\‘
INFTP=\‘cat \${EXPDIR}/saveftp.\${EXP}\‘
TARFILE=\‘cat \${EXPDIR}/savetar.\${EXP}\‘
#
YY=\$INYEAR
if [ \${INYEAR} -lt 10 ]
then
YY=0\${YY}
fi
MM=\$INMON
if [ \${INMON} -lt 10 ]
then
MM=0\${MM}
fi
#
# make namelist
#
\$EXPDIR/NAMLIST.exe 1
cp \${EXPDIR}/land_namelist land_namelist
cp \${EXPDIR}/sea_namelist sea_namelist
cp \${EXPDIR}/ice_namelist ice_namelist
cp \${EXPDIR}/ocean_namelist ocean_namelist
```

```
#
cp \$EXPDIR/\${EXP}RES puma_restart
cp \$EXPDIR/\${EXP}LANDRES land_restart
cp \$EXPDIR/\${EXP}SEARES sea_restart
#
model.x > \${EXPDIR}/\${EXP}PROUT_\${YY}\${MM}
#
# history and restart saved for further diagnostics
#
mv puma_output \${EXP}PUMA_\${YY}\${MM}
#
if [ \${INFTP} -eq 1 ]
then
TARFILE=\${EXP}TAR_\${YY}\${MM}
echo \${TARFILE} > \${EXPDIR}/savetar.\${EXP}
tar -cf \${EXPDIR}/\${TARFILE} \${EXP}PUMA_\${YY}\${MM}
else
tar -rf \${EXPDIR}/\${TARFILE} \${EXP}PUMA_\${YY}\${MM}
fi
mv puma_restart \$EXPDIR/\${EXP}RES
mv land_restart \$EXPDIR/\${EXP}LANDRES
mv sea_restart \$EXPDIR/\${EXP}SEARES
rm \${EXP}PUMA_\${YY}\${MM}
#
OUTMON=\`expr \${INMON} + 1\`
OUTDAY=\`expr \${INDAY} + 30\`
OUTYEAR=\${INYEAR}
OUTFTP=\`expr \${INFTP} + 1\`
#
if [ \${OUTMON} -eq 13 ]
then
OUTMON=1
OUTYEAR=\`expr \${OUTYEAR} + 1\`
fi
#
echo \${OUTYEAR} > \${EXPDIR}/saveyear.\${EXP}
echo \${OUTMON} > \${EXPDIR}/savemon.\${EXP}
echo \${OUTDAY} > \${EXPDIR}/saveday.\${EXP}
#
#################################################################
#
if [ \${OUTFTP} -gt \${FTPINT} ]
then
#
cd \${EXPDIR}
OUTFTP=1
```

```
echo \${OUTFTP} > \${EXPDIR}/saveftp.\${EXP}
tar -rf \${EXPDIR}/\${TARFILE} \${EXP}RES
tar -rf \${EXPDIR}/\${TARFILE} \${EXP}LANDRES
tar -rf \${EXPDIR}/\${TARFILE} \${EXP}SEARES
tar -rf \${EXPDIR}/\${TARFILE} NAMLIST.exe
tar -rf \${EXPDIR}/\${TARFILE} land_namelist
tar -rf \${EXPDIR}/\${TARFILE} sea_namelist
tar -rf \${EXPDIR}/\${TARFILE} ice_namelist
tar -rf \${EXPDIR}/\${TARFILE} ocean_namelist
tar -rf \${EXPDIR}/\${TARFILE} saveday.\${EXP}
tar -rf \${EXPDIR}/\${TARFILE} savemon.\${EXP}
tar -rf \${EXPDIR}/\${TARFILE} saveyear.\${EXP}
tar -rf \${EXPDIR}/\${TARFILE} savetar.\${EXP}
tar -rf \${EXPDIR}/\${TARFILE} saveftp.\${EXP}
tar -rf \${EXPDIR}/\${TARFILE} model.x
tar -rf \${EXPDIR}/\${TARFILE} runall
mv \${EXPDIR}/\${TARFILE} \${DATADIR}/\${TARFILE}to\${YY}\${MM}
if [ \${SCHAUER} -eq 1 ]
then
cat > \${EXPDIR}/put_schauer\${YY}\${MM} << EOP
set -ex
cd \${EXPDIR}
/home/larry/utils/uput2 \${DATADIR}/\${TARFILE}to\${YY}\${MM} \${SCHAUERDIR}/\${TARFI
rm \${DATADIR}/\${TARFILE}to\${YY}\${MM}
EOP
chmod u+x \${EXPDIR}/put_schauer\${YY}\${MM}
\${EXPDIR}/put_schauer\${YY}\${MM} > \${EXPDIR}/put_\${YY}\${MM} 2>&1 &
fi
cd \$TMPDIR
fi
#
echo \${OUTFTP} > \${EXPDIR}/saveftp.\${EXP}
#
if [ \${OUTYEAR} -gt \${LASTYEAR} ]
then
cd \${EXPDIR}
rm -r \${TMPDIR}
exit
fi
#
exit
EOR
#
cd $EXPDIR
chmod u+x runall
runall
```

```
#
etime=‘date‘
echo " "
echo "==> started, $stime"
echo " "
echo "==> done,    $etime"
echo " "
#
exit
```

Chapter 4

Parallel Program Execution

4.1 Concept

The **Planet Simulator** is coded for parallel execution on computers with multiple CPU's or networked machines. The implementation uses MPI (Message Passage Interface), that is available for nearly every operating system `http://www.mcs.anl.gov/mpi`.

In order to avoid maintaining two sets of source code for the parallel and the single CPU version, all calls to the MPI routines are encapsulated into a module. Users, that want to compile and execute the parallel version use the module **mpimod.f90** and the commands **mpif90** for compiling and **mpirun** for running.

If MPI is not implemented or the single CPU version is sufficient, the module **mpimod_dummy.f90** is used instead of mpimod.f90. Also remove or comment the line:

```
!     use mpi
```

and set the number of processors to 1:

```
      parameter(NPRO = 1)
```

4.2 Parallelization in Gridpoint Domain

The data arrays in gridpoint domain are either three-dimensional e.g. gt(NLON, NLAT, NLEV) referring to an array organized after longitudes, latitudes and levels, or two-dimensional, e.g. gp(NLON, NLAT). The code is organized such, that there are no dependencies in latitudinal direction, while in gridpoint domain. Such dependencies are resolved during the Legendre-Transformations. So the the partitioning of the data is done in latitudes. The program can use as many CPU's as latitudes with the extreme of every CPU doing the computations for a single latitude. There is the restriction however, that the number of latitudes (NLAT) divided by the number of processors (NPRO), giving the number of latitudes per process (NLPP) must have zero remainder. E.g. A T31 resolution uses $NLAT = 48$. Possible values for NPRO are then 1, 2, 3, 4, 6, 8, 12, 16, 24, and 48.

All loops dealing with a latitudinal index look like:

```
      do jlat = 1 , NLPP
         ....
      enddo
```

There are, however, many subroutines, with the most prominent called **calcgp**, that can fuse latitudinal and longitudinal indices. In all these cases the dimension NHOR is used. NHOR is defined as: $NHOR = NLON * NLPP$ in the pumamod - module. The typical gridpoint loop that looks like:

```
do jlat = 1 , NLPP
   do jlon = 1 , NLON
      gp(jlon,jlat) = ...
   enddo
enddo
```

is then replaced by the faster executing loop:

```
do jhor = 1 , NHOR
   gp(jhor) = ...
enddo
```

4.3 Parallelization in Spectral Domain

The number of coefficients in spectral domain (NRSP) is divided by the number of processes (NPRO) giving the number of coefficients per process (NSPP). The number is rounded up to the next integer and the last process may get some additional dummy elements, if there is a remainder in the division operation.

All loops in spectral domain are organized like:

```
do jsp = 1 , NSPP
   sp(jsp) = ...
enddo
```

4.4 Synchronization points

All processes must communicate and have therefore to be synchronized at following events:

- Legendre-Transformation: This involves changing from latitudinal partitioning to spectral partitioning and such some gather and scatter operations.
- Inverse Legendre-Transformation: The partitioning changes from spectral to latitudinal by using gather, broadcast, and scatter operations.
- Input-Output: All read and write operations must be done only by the root process, who gathers and broadcasts or scatters the information as desired. Code that is to be executed by the root process exclusively is written like:

  ```
  if (mypid == NROOT) then
     ...
  endif
  ```

  NROOT is typically 0 in MPI implementations, mypid (My process identification) is assigned by MPI.

4.5 Source code

It needs some discipline in order to maintain parallel code. Here are the most important rules for changing or adding code to the **Planet Simulator**:

- Adding namelist parameters: All namelist parameters must be broadcasted after reading the namelist. (Subroutines mpbci, mpbcr, mpbcin, mpbcrn)
- Adding scalar variables and arrays: Global variables must be defined in a module header and initialized.
- Initialization code: Initialization code, that contains dependencies on latitude or spectral modes must be done by the root process only and then scattered from there to all child processes.
- Array dimensions and loop limits: Always use parameter constants as defined in pumamod.f90 for array dimensions and loop limits.
- Testing: After significant code changes the program should be tested in single and in multi-CPU configuration. The results of a single CPU run is usually not exactly the same as the result of a multi-CPU run due to effects in rounding. But the results should show only small differences during the first timesteps.
- Synchronization points The code is optimzed for parallel execution and minimizes therefore communication overhead. The necessary communication code is grouped around the Legendre-transformations. If more scatter/gather operations or other communication routines are to be added, they should be placed just before or after the execution of the calls to the Legendre-Transformation. Any other place would degrade the overall performance in introducing additional process synchronization.

Chapter 5

Graphical User Interface

5.1 Graphical user interface (GUI)

:

The Planet Simulator may be used in the traditional fashion, with shell scripts, batch jobs, and network queuing systems. This is acceptable for long running simulations on complex machines and number-crunchers, like vector- computers, massive-parallel-computers and workstation clusters. There is now, however, a much more convenient method by using a graphical user interface (GUI) for model setup with parameter configurations and for interaction between user and model.

The Planet Simulator is configured and setup by the first GUI module named MoSt (Model Starter, screenshot in figure **NN**). MoSt is the fastest way to get the model running. It gives access to the most important parameters of the model preset to the most frequently used values. The model can be started with a mouse click on the button labelled "Save & Run" either with the standard paramater setting or after editing some of the parameters in the MoSt window. Some parameters, like horizontal and vertical resolution, or the number of processors, require the building (compile, link and load) of new executables. MoSt achieves this by generating and executing build scripts, that perform the necessary code changes and create the required executable. Other parameters define startup- and boundary conditions or settings for parameterisations. They can be edited in MoSt and, after a check for correct range and consistency with other parameters, are written to the model's namelist file.

Depending on all settings MoSt generates a runscript for the simulation. The user has the choice of leaving MoSt and continue with the simulation under control of a GUI right away, or to exit MoSt with the scripts prepared to run. The second alternative is useful for users, who want to modify the setup beyond the scope of MoSt or want to run the Planet Simulator without GUI.

The GUI for running the Planet Simulator (screenshot in figure **NN**) has two main purposes. The first one is to display model arrays in suitable representations. Current implementations are:

- Zonal mean cross sections
- Horizontal global fields in cylinder projection
- Horizontal global fields in polar projection

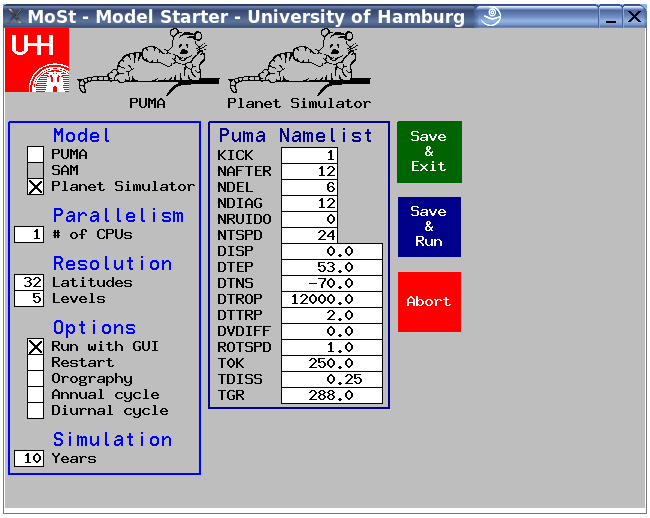


Figure 5.1: Screenshot of Model Starter (MoSt)

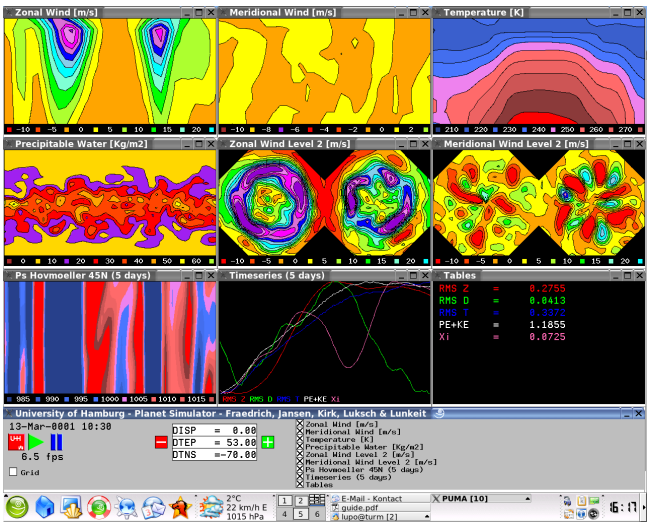


Figure 5.2: Screenshot of Graphical User Interface (GUI)

- Time-longitude (Hovmoeller) diagrams
- Time series
- Numerical values

The second purpose is the interaction part of the GUI, which allows the user to change selected model variables during the model run. It is not necessary, though possible, to pause the model while changing variables. Changes to model variables are, of course, monitored in the outputfile and checked by GUI for the appropriate range of values and maximum possible change per timestep because, otherwise, a rapid parameter change or a choice of values beyond the normal range may blow up the model.

All model variables, which are candidates for the display or interactive changes, have a special code to communicate with the Planet Simulator. The experienced modeller can add new code for more variables using the existing communication code as template. Thus all model fields or even fields received via coupling with other models can be put on the GUI display.

Both, MoSt and GUI are implemented using the Xlib (X11R5), which is a library of routines for graphics and event communication. As this library is part of every UNIX/Linux operating system and base of all desktop environments, there is no need to install additional software for running MoSt and GUI. Another important property of Xlib is the full network transparency. The display of MoSt and GUI is not locked to the machine running the programs or the model. In fact, the best performance is obtained in running the Planet Simulator on two or four CPUs of a remote server while displaying the GUI on the user's workstation. In summarizing, the MoSt and GUI programs automate many tedious tasks, minimize the time to become familiar with the Planet Simulator, and make debugging and parameter tuning much easier. More kinds of presentations, coordinate projections and interactivity are being developed. A graphical preprocessor with editor for boundary conditions and a graphical postprocessor are future expansions to build an almost complete environment for modellers.

Chapter 6

Postprocessor Pumaburner

6.1 Introduction

The **Pumaburner** is a postprocessor for the **Planet Simulator** and the **PUMA** model family. It's the only interface between *raw* model data output and diagnostics, graphics, and user software.

The output data of the **Planet Simulator** are stored as packed binary (16 bit) values using the model representation. Prognostic variables like temperature, divergence, vorticity, pressure, and humidity are stored as coefficients of spherical harmonics on σ levels. Variables like radiation, precipitation, evaporation, clouds, and other fields of the parameterization package are stored on Gaussian grids.

The tasks of the **Pumaburner** are:

- Unpack the *raw* data to full real representation.
- Transform variables from the model's representation to a user selectable format, e.g. grids, zonal mean cross sections, fourier coefficients.
- Calculate diagnostic variables, like vertical velocity, geopotential height, wind components, etc.
- Transfrom variables from σ levels to user selectable pressure levels.
- Compute monthly means and standard deviations.
- Write selected data either in SERVICE, GRIB, or NetCDF format for further processing.

6.2 Usage

```
pumaburn4 [options] InputFile OutputFile <namelist >printout
     option -h : help (this output)
     option -c : print available codes and names
     option -d : debug mode (verbose output)
     option -g : Grib   output (override namelist option)
     option -n : NetCDF output (override namelist option)
```

```
 option -m : Mean=1 output (override namelist option)
 InputFile : Planet Simulator or PUMA data file
OutputFile : GRIB, SERVICE, or NetCDF format file
  namelist : redirected <stdin>
  printout : redirected <stdout>
```

6.3 Namelist

The namelist values control the selection, coordinate system and output format of the post-processed variables. Names and values are not case sensitive. You can assign values to the following names:

Name	Def.	Type	Description	Example
HTYPE	S	char	Horizontal type	HTYPE=G
VTYPE	S	char	Vertical type	VTYPE=P
MODLEV	0	int	Model levels	MODLEV=2,3,4
hPa	0	real	Pressure levels	hPa=500,1000
CODE	0	int	ECMWF field code	CODE=130,152
GRIB	0	int	GRIB output selector	GRIB=1
NETCDF	0	int	NetCDF output selector	NETCDF=1
MEAN	1	int	Compute monthly means	MEAN=0
HHMM	1	int	Time format in Service format	HHMM=0
HEAD7	0	int	User parameter	HEAD7=0815
MARS	0	int	Use constants for planet Mars	MARS=1
MULTI	0	int	Process multiple input files	MULTI=12

6.4 HTYPE

HTYPE accepts the first character of the following string. Following settings are equivalent: HTYPE = S, HTYPE=Spherical Harmonics HTYPE = Something. Blanks and the equal-sign are optional.
Possible Values are:

Setting	Description	Dimension for T21 resolution
HTYPE = S	Spherical Harmonics	(506):(22 * 23 coefficients)
HTYPE = F	Fourier Coefficients	(32,42):(latitudes,wavenumber)
HTYPE = Z	Zonal Means	(32,levels):(latitudes,levels)
HTYPE = G	Gauss Grid	(64,32):(longitudes,latitudes)

6.5 VTYPE

VTYPE accepts the first character of the following string. Following settings are equivalent: VTYPE = S, VTYPE=Sigma VTYPE = Super. Blanks and the equal-sign are optional.
Possible Values are:

Setting	Description	Remark
VTYPE = S	Sigma (model) levels	Some derived variables are not available
VTYPE = P	Pressure levels	Interpolation to pressure levels

6.6 MODLEV

MODLEV is used in combination with **VTYPE = S**. If VTYPE is not set to Sigma, the contents of MODLEV are ignored. MODLEV is an integer array that can get as many values as there are levels in the model output. The levels are numbered from top of the atmosphere to the bottom. The number of levels and the corresponding sigma values are listed in the pumaburner printout. The outputfile orders the level according to the MODLEV values. MODLEV=1,2,3,4,5 produces an output file of five model levels sorted from top to bottom, while MODLEV=5,4,3,2,1 sorts them from bottom to top.

6.7 hPa

hPa is used in combination with **VTYPE = P**. If VTYPE is not set to Pressure, the contents of hPa are ignored. hPa is a real array that accepts pressure values with the units hectoPascal or millibar. All output variables will be interpolated to the selected pressure levels. There is no extrapolation on the top of the atmosphere. For pressure values, that are lower than that of the model's top level, the top level value of the variable is taken. The variables temperature and geopotential height are extrapolated if the selected pressure is higher than the surface pressure. All other variables are set to the value of the lowest mode level for this case. The outputfile contains the levels in the same order as set in hPa. Example: hpa = 100,300,500,700,850,900,1000.

6.8 MEAN

MEAN can be used to compute montly means and/or deviations. The Pumaburner reads date and time information from the model file and handles different lengths of months and output intervals correctly.

Setting	Description
MEAN = 0	Do no averaging - all terms are processed.
MEAN = 1	Compute and write monthly mean fields. Not for spherical harmonics, Fourier coefficients or zonal means on sigma levels.
MEAN = 2	Compute and write monthly deviations. Not for spherical harmonics, Fourier coefficients or zonal means on sigma levels. Deviations are not available for NetCDF output.
MEAN = 3	Combination of MEAN=1 and MEAN=2. Each mean field is followed by a deviation field with an identical header record. Not for spherical harmonics, Fourier coefficients or zonal means on sigma levels.

6.9 Format of output data

The **pumaburner** supports three different output formats:

- **GRIB** (GRIdded Binary) WMO standard for gridded data.

- **NetCDF** (Network Common Data Format)
- **Service** Format for user readable data (see below).

For more detailed descriptions see for example:
`http://www.nws.noaa.gov/om/ord/iob/NOAAPORT/resources/`

Setting	Description	
GRIB = 1	NetCDF = 0	The output file is written GRIB format. This option can be used only for HTYPE=Spherical Harmonics or HTYPE=Gauss Grid.
GRIB = 0	NetCDF = 1	The output file is written in NetCDF format. This option can be used for HTYPE=Gauss Grid only.
GRIB = 0	NetCDF = 0	The output file is written in Service format. This is the preferred format for user programs. For a detailed description see the following section.
GRIB = 1	NetCDF = 1	Illegal combination.

6.10 SERVICE format

The SERVICE format uses the following structure: The whole file consists of pairs of header records and data records. The header record is an integer array of 8 elements.

```
    head(1) = ECMWF field code
    head(2) = modellevel or pressure in [Pa]
    head(3) = date  [yymmdd]  (yymm00 for monthly means)
    head(4) = time  [hhmm]  or [hh] for HHMM=0
    head(5) = 1. dimension of data array
    head(6) = 2. dimension of data array
    head(7) = may be set with the parameter HEAD7
    head(8) = experiment number (extracted from filename)

    Example for reading the SERVICE format (GRIB=0 , NETCDF=0)

    INTEGER HEAD(8)
    REAL    FIELD(64,32)     ! dimensions for T21 grids
    READ (10,ERR=888,END=999) HEAD
    READ (10,ERR=888,END=999) FIELD
    ....
888 STOP 'I/O ERR'
999 STOP 'EOF'
    ....
```

6.11 HHMM

Setting	Description
HHMM = 0	head(4) shows the time in hours (HH).
HHMM = 1	head(4) shows the time in hours and minutes (HHMM).

6.12 HEAD7

The 7th. element of the header is reserved for the user. It may be used for experiment numbers, flags or anything else. Setting HEAD7 to a number exports this number to every header record in the output file (SERVICE format only).

6.13 MARS

This parameter is used for processing simulations of the Mars atmosphere. Setting MARS=1 switches gravity, gas constant and planet radius to the correct values for the planet Mars.

6.14 MULTI

The parameter MULTI can bes used to process a series of input data within one run of the pumaburner. Setting MULTI to a number (n) tells the pumaburner to procees (n) input files. The input files must follow one of the following two rules:

- YYMM rule: The last four characters of the filename contain the data in the form YYMM.
- .NNN rule: The last four characters of the filename consist of a dot followed ny a 3-digit sequence number.

```
Examples:

Namelist contains MULTI=3
Command: pumaburn <namelist >printout run.005 out
pumaburn processes the files <run.005> <run.006> <run.007>

Namelist contains MULTI=4
Command: pumaburn <namelist >printout exp0211 out
pumaburn processes the files <exp0211> <exp0212> <exp0301> <exp0302>
```

6.15 Namelist example

```
VTYPE  = Pressure
HTYPE  = Grid
CODE   = 130,131,132
hPa    = 200,500,700,850,1000
MEAN   = 0
GRIB   = 0
NETCDF = 0
```

This namelist will write Temperature(130), u(130) and v(131) on pressure levels 200hPa, 500hPa, 700hPa, 850hPa and 1000hPa. The output interval is the same as found on the model data, e.g. every 12 or every 6 hours (MEAN=0). The output format is SERVICE format.

6.16 Troubleshooting

If the pumaburner reports an error or doesn't produce the expected results, try the following:

- Check your namelist, especially for invalid codes, types and levels.
- Run the pumaburner in debug-mode by using the option -d. Example:

  ```
  pumaburn <namelist >printout -d data.in data.out
  ```

  This will print out some details like parameters and memory allocation during the run. The additional information may help to detect the problem.
- Not all combinations of HTYPE, VTYPE, and CODE are valid. Try to use HTYPE=Grid and VTYPE=Pressure before switching to exotic parameter combinations.

Chapter 7

Graphics

7.1 Grads

In this section, visualisation using the graphics package GrADS is described. A useful Internet site for reference and installation instructions is

<http://grads.iges.org/grads/grads.html>.

Latest versions of GrADS can handle data in NETCDF format (via the command sdfopen), GRIB, HDF-SDS, and in its native binary format. The native format can conveniently be derived from SERVICE format. In the following it is assumed that the PUMA output has been converted to SERVICE format with the pumaburner and the resulting file is called puma.srv. Monthly mean data is either obtained directly from the pumaburner (namelist parameter MEAN=1, see section 6) or via a PINGO command:

```
srv monmeans puma.srv puma_m.srv
```

Information on the PINGO package can be found in DKRZ report 11 at

<http://www.mad.zmaw.de/Pingo/repdl.html>.

The SERVICE file has to be converted to GrADS's native format by the command:

```
srv2gra puma_m.srv
```

which results in the files puma_m.gra and puma_m.ctl. The first file contains the data, the latter one information on the grid, time steps, and variable names. The program srv2gra is one of the postprocessing tools available at

<http://puma.dkrz.de/puma/download/map/>.

If you chose to compile it yourself, please read the comments in the first few lines of the program text. Sometimes the srv2gra tool has difficulties to calculate an appropriate time increment from the date headers of the data records, so you should check this. In this example the file puma_m.ctl should look like this:

```
DSET ^puma_m.gra
UNDEF 9e+09
XDEF     64 LINEAR   0.0000   5.6250
OPTIONS YREV
YDEF     32 LEVELS
  -85.7606  -80.2688  -74.7445  -69.2130  -63.6786  -58.1430  -52.6065  -47.0696
  -41.5325  -35.9951  -30.4576  -24.9199  -19.3822  -13.8445   -8.3067   -2.7689
    2.7689    8.3067   13.8445   19.3822   24.9199   30.4576   35.9951   41.5325
   47.0696   52.6065   58.1430   63.6786   69.2130   74.7445   80.2688   85.7606
ZDEF  1 LINEAR 1 1
TDEF 12 LINEAR 00:00Z01jan0001        1mo
VARS  3
c139     0 99    139     0     0
c151     0 99    151     0     0
c175     0 99    175     0     0
ENDVARS
```

Here, the line starting with `TDEF` ends with `1mo`, since we are handling monthly mean data. When the `PUMA` output is used without averaging, this should correspond to the output interval given by the `nafter` variable used in the `namelist` of your `PUMA` run (see section **??**). The number of variables depends on how the pumaburner was called. In this example, only 3 variables were processed, i.e. the surface temperature (`c139`), the sea level pressure (`c151`) and the albedo (`c175`; refer to appendix **??** for a list of codes).
The GrADS program is started by typing `grads` in a terminal window. Then, data is visualised either by typing commands line-by-line, or, preferably, by using scripts. The following script, called `tglob.gs`, displays the monthly mean surface temperature:

```
# tglob.gs
function pass(m)
'reinit'
'open puma_m'
'enable print print.mf'
'set t 'm
'c'
'set gxout shaded'
'd (c139-273.16)'
'cbar.gs'
'set gxout contour'
'd (c139-273.16)'
'draw title Surface Temperature (deg C) month 'm
'print'
'disable print'
'!gxps -i print.mf -o tglob'm'.ps'
```

The variable `m` at the beginning of the script defines the month which should be displayed. It is passed from the terminal with the script call. Note that in this line, no quotation marks are present, since only `GrADS` specific commands are framed by quotation marks. Script commands,

like variable definitions, if-clauses etc. are used without quotation marks. The script is executed by typing its name without the ending and the number of the month to be shown. For example, `tglob 7` displays the monthly mean surface temperature in July. The resulting output file is called `tglob7.ps`.

The following script `thh` displays the time dependent surface temperature of Hamburg. Here, two variables are passed to `GrADS`, the first and last day to plot (note that here, the file `puma.gra` is opened, which contains data on a daily basis). The call `thh 91 180` displays the surface temperature of Hamburg for the spring season from April 1st to June 30th.

```
# thh.gs
function pass(d1 d2)
'reinit'
'open puma'
'enable print print.mf'
'set lat 53'
'set lon 10'
'set t 'd1' 'd2
'c'
'd (c139-273.16)'
'draw title Surface Temperature (deg C) in Hamburg'
'print'
'disable print'
'!gxps -i print.mf -o thh.ps'
```

It is possible to have more than one figure in a plot, which is illustrated in the following script. It plots seasonal means of the sea level pressure. The data file is prepared like this:

```
srv selcode,151 puma.srv slp.srv
srv seasmean slp.srv slp_sm.srv
srv2gra slp_sm.srv
```

The commands `set vpage` sets virtual pages inside the graphic window. The full window is 11 inch wide and 8.5 inch high, so `set vpage 0 5.5 4.25 8.5` defines the upper left corner. If `setlevs=1` is specified, the pressure levels as given are used. Otherwise, `GrADS` defines contour levels depending on the data set.

```
# slp_sm.gs
setlevs=1
'reinit'
'open slp_sm'
'enable print print.mf'
'c'
'set vpage 0 5.5 4.25 8.5'
'set gxout contour'
if (setlevs=1)
'set clevs 990 995 1000 1005 1010 1015 1020'
```

```
endif
'set ccols 1'
'set grads off'
'set t 1'
'd c151/100'
'draw title SLP [hPa] yr 'ny' DJF'
'set vpage 5.5 11 4.25 8.5'
'set gxout contour'
if (setlevs=1)
'set clevs 990 995 1000 1005 1010 1015 1020'
endif
'set ccols 1'
'set grads off'
'set t 2'
'd c151/100'
'draw title yr 'ny' MAM'
'set vpage 0 5.5 0 4.25'
'set gxout contour'
if (setlevs=1)
'set clevs 990 995 1000 1005 1010 1015 1020'
endif
'set ccols 1'
'set grads off'
'set t 3'
'd c151/100'
'draw title yr 'ny' JJA'
'set vpage 5.5 11 0 4.25'
'set gxout contour'
if (setlevs=1)
'set clevs 990 995 1000 1005 1010 1015 1020'
endif
'set ccols 1'
'set grads off'
'set t 4'
'd c151/100'
'draw title yr 'ny' SON'
'print'
'disable print'
'!gxps -c -i print.mf -o slp_sm.ps'
```

7.2 Vis5D

"Vis5D is a system for interactive visualization of large 5-D gridded data sets such as those produced by numerical weather models. One can make isosurfaces, contour line slices, colored slices, volume renderings, etc of data in a 3-D grid, then rotate

and animate the images in real time. There's also a feature for wind trajectory tracing, a way to make text annotations for publications, support for interactive data analysis, etc."

from the Vis5D home page,
`http://www.ssec.wisc.edu/~billh/vis5d.html`

This powerful visualisation tool together with its documentation is available through the above home page. Vis5D uses its own data format which makes it necessary to transform your data. Depending on their format and the flowchart on `http://puma.dkrz.de/puma/download/map/` you have the following choices: If

- *your data is raw* `PUMA` *output,*
  you need to process it with the `pumaburner` postprocessor (see section 6) in order to transform it to either `NETCDF` (option `-n` or namelist parameter `NETCDF=1`) or `GRIB` (option `-g` or namelist parameter `GRIB=1`) and proceed from there.

- *your data is in* `SERVICE` *format,*
  you need to convert it to either `GRIB`, for instance with the `PINGO`s:

  `grb copy2 data.srv data_with_grib_metainfo.grb output.grb`,

  or `NETCDF`, using the program `puma2cdf`, which is available with the `PUMA` postprocessing tools. Despite of its name this program cannot process raw `PUMA` output but takes `SERVICE` format as input. It can as well be called as `srv2cdf` which changes its behaviour: oddities of model output such as the existence of February, 30^{th} are then no longer removed. Once the format is changed proceed from there.

- *your data is in* `NETCDF` *format,*
  it can easily transformed to `Vis5D`'s native format by means of the program `cdf2v2d`, which is available with the `PUMA` postprocessing tools.

- *your data is in* `GRIB` *format,*
  you can find a transformation tool named `Grib2V5d` at
  `<http://grib2v5d.sourceforge.net>` which offers various practical features.

Once the conversion to `Vis5D`'s native format is achieved please follow the instructions from the `Vis5D` documentation or, if `Vis5D` is already installed on your system, try finding your own way by typing:

```
vis5d my_data.v5d
```

Appendix A

List of Constants and Symbols

A.0.1 List of Constants and Symbols

Symbol	Definition	Value	Unit
a	earth radius	$6371{\cdot}10^3$	m
A	$= D + \vec{V} \cdot \nabla \ln p_s$		–
$\mathcal{A}$	absorptivity/emissivity		–
$\mathcal{A}_S$	surface emissivity		–
$B(T)$	Planck function		$\mathrm{Wm^{-2}}$
cc	cloud cover		–
C_{char}	Charnock constant	0.018	–
C_h	transfer coefficient for heat		–
C_m	drag coefficient for momentum		–
c_p	specific heat of moist air at constant pressure		$\mathrm{J\,kg^{-1}\,K^{-1}}$
c_{pd}	specific heat of dry air at constant pressure	1005.46	$\mathrm{J\,kg^{-1}\,K^{-1}}$
c_{pv}	specific heat of water vapor at constant pressure	1869.46	$\mathrm{J\,kg^{-1}\,K^{-1}}$
c_{p_i}	specific heat of sea ice	2070	$\mathrm{W\,s\,kg^{-1}\,K^{-1}}$
c_{p_s}	specific heat of snow	2090	$\mathrm{W\,s\,kg^{-1}\,K^{-1}}$
c_{p_w}	specific heat of sea water	4180	$\mathrm{W\,s\,kg^{-1}\,K^{-1}}$
c_w	coefficient for the deep ocean heat flux	4	$\mathrm{W\,m^{-2}\,K^{-1}}$
C_w	wetness factor		–
D	scaled divergence		–
E	evaporation		$\mathrm{m\,s^{-1}}$
E_0	extrateristical solar flux density		$\mathrm{W\,m^{-2}}$
f	Coriolis parameter $=: 2\Omega \sin\varphi$		$\mathrm{s^{-1}}$
F_p	tendency of the first moment$=: \frac{dR_1}{dt}$		$\mathrm{K\,m^2\,s^{-1}}$
F_q	tendency of the zeroth moment$=: \frac{dR_0}{dt}$		$\mathrm{K\,m\,s^{-1}}$
F_q	surface moisture flux		$\mathrm{kg\,m^{-2}\,s^{-1}}$
F_T	surface sensible heat flux		$\mathrm{W\,m^{-2}}$
F_u	surface zonal wind stress		Pa
F_v	surface meridional wind stress		Pa
F^{LW}	long wave radiation flux density		$\mathrm{Wm^{-2}}$
F^{SW}	short wave radiation flux density		$\mathrm{Wm^{-2}}$
g	gravitational acceleration	9.81	$\mathrm{m^{-2}}$
h_{mix}	mixed layer depth		m
h_{mix_c}	climatological mixed layer depth		m
H_q	effective mixed layer depth $=: \frac{R_0}{T_{mix}-T_{ref}}$		m
H_p	reduced center of gravity $=: \frac{R_1}{R_0}$		m
J_q	vertical turbulent moisture flux		$\mathrm{kg\,m^{-2}\,s^{-1}}$

L_f	latent heat of fusion $= L_s - L_v$	$3.28{\cdot}10^5$	$\mathrm{J\,kg^{-1}}$
l_h	mixing length for heat		m
l_m	mixing length for momentum		m
L_s	latent heat of sublimation	$2.8345{\cdot}10^6$	$\mathrm{J\,kg^{-1}}$
L_v	latent heat of vapourization	$2.5008{\cdot}10^6$	$\mathrm{J\,kg^{-1}}$
P_c	convective precipitation		$\mathrm{ms^{-1}}$
P_l	large scale precipitation		$\mathrm{ms^{-1}}$
$P_n^m(\mu)$	associated Legendre function of the first kind		–
p	pressure		Pa
p_S	surface pressure		Pa
p_s	scaled surface pressure		–
q	specific humidity		$\mathrm{kg\,kg^{-1}}$
Q	total heat flux through sea ice		$\mathrm{W\,m^{-2}}$
$\tilde{Q}$	flux correction heat flux through sea ice		$\mathrm{W\,m^{-2}}$
Q_a	total atmospheric heat flux		$\mathrm{W\,m^{-2}}$
Q_c	conductive heat flux through sea ice		$\mathrm{W\,m^{-2}}$
Q_f	heat flux available for freezing sea ice		$\mathrm{W\,m^{-2}}$
Q_g	heat flux into the soil		$\mathrm{Wm^{-2}}$
Q_m	snow melt heat flux		$\mathrm{Wm^{-2}}$
Q_o	oceanic heat flux		$\mathrm{W\,m^{-2}}$
q_S	surface specific humidity		$\mathrm{kg\,kg^{-1}}$
q_{sat}	saturation specific humidity		$\mathrm{kg\,kg^{-1}}$
$\mathcal{R}$	refexivity/albedo		–
$\mathcal{R}_S$	surface albedo		–
R_d	gas constant for dry air	287.05	$\mathrm{J\,kg^{-1}\,K^{-1}}$
R_l	surface long wave radiation		$\mathrm{W\,m^{-2}}$
R_s	surface short wave radiation		$\mathrm{W\,m^{-2}}$
R_v	gas constant for water vapor	461.51	$\mathrm{J\,kg^{-1}\,K^{-1}}$
R_0	zeroth moment of the temperature distribution		$\mathrm{K\,m}$
R_1	first moment of the temperature distribution		$\mathrm{K\,m^2}$
Ri	Richardson number		–
S_w	salinity of sea water	34.7	psu
t	time		s
t	scaled time step		–
$\mathcal{T}$	transmissivity		–
T	temperature		K
T'	temperature anomaly $=: T - T_0$		–
T_d	deep ocean temperature (at 400m)		K
T_i	sea ice surface temperature		K
T_f	freezing temperature	271.25	K
T_s	surface temperature		K
T_{sea}	sea surface temperature		K
T_{melt}	melting point	273.16	K
T_{mix}	mixed layer temperature		K
T_{mix_c}	climatological mixed layer temperature		K
T_{ref}	asymptotic reference temperature		K

T_w	oceanic temperature profile		K
T_0	reference temperature profile	250.0	K
U	scaled zonal wind $=: u \cdot \cos\varphi$		–
u	zonal wind		$\mathrm{m\,s^{-1}}$
u_*	friction velocity		$\mathrm{m\,s^{-1}}$
V	scaled meridional wind $=: v \cdot \cos\varphi$		–
v	meridional wind		$\mathrm{m\,s^{-1}}$
$\vec{v}$	horizontal wind vector		$\mathrm{m\,s^{-1}}$
W_L	cloud liquid water path		$\mathrm{gm^2}$
W_{snow}	mass of snow water		kg
W_{soil}	soil water		m
z	height		m
z_0	roughness length		m
Δt	time increment		s
Δz	height increment		m
α	thermal expansion coefficient $\frac{1}{\rho}\frac{d\rho}{dT}$	$2.41 \cdot 10^{-4}$	$\mathrm{K^{-1}}$
β	back scattering coefficient		–
β_d	diffusivity factor	1.66	–
ζ	scaled vorticity		–
θ	potential temperature		K
κ	R_d/C_{pd}		–
$\bar{\kappa}$	mean heat conductivity in ice and snow		$\mathrm{W\,m^{-1}\,K^{-1}}$
κ_i	heat conductivity in ice	2.03	$\mathrm{W\,m^{-1}\,K^{-1}}$
κ_s	heat conductivity in snow	0.31	$\mathrm{W\,m^{-1}\,K^{-1}}$
λ_h	asymptotic mixing length for heat		m
λ_m	asymptotic mixing length for momentum		m
λ	longitude		–
μ	$\sin\varphi$		–
μ_0	cosine of the solar zenith angle		–
ρ	density of air		$\mathrm{kg\,m^{-3}}$
ρ_i	density of sea ice	920	$\mathrm{kg\,m^{-3}}$
ρ_s	density of snow	330	$\mathrm{kg\,m^{-3}}$
ρ_w	density of sea water	1030	$\mathrm{kg\,m^{-3}}$
ρ_o	density of fresh water	1000	$\mathrm{kg\,m^{-3}}$
σ	normalized pressure coordinate $=: p/p_s$		–
$\dot{\sigma}$	vertical velocity in σ system		–
σ_{SB}	Stefan-Bolzmann constant	$5.67 \cdot 10^{-8}$	$\mathrm{Wm^{-2}K^{-4}}$
τ_N	cloud optical depth		–
τ_F	time scale for RF		–
τ_R	time scale for NC		–
τ_T	time scale for temperature flux correction		s
τ_h	time scale for depth flux correction		s
ϕ	geopotential height $:= g \cdot z$		$\mathrm{m^2\,s^{-2}}$
ϕ	scaled geopotential height		–
φ	latitude		–
χ	scaled velocity potential		–

ψ	scaled streamfunction		–
Ω	angular velocity of the earth	$7.292 \cdot 10^{-5}$	s^{-1}
$\tilde{\omega_0}$	single scattering albedo		–

Appendix B

Puma Codes

Codes available from PUMA-burner (adapted from ECHAM)

Code	Levels	Type	Variable	Unit
110	1	g	mixed layer depth	m
129	1	s	surface geopotential	m^2/s^2
130	NLEV	s	temperature	K
131	NLEV	c	u-velocity	m/s
132	NLEV	c	v-velocity	m/s
133	NLEV	s	specific humidity	kg/kg
135	NLEV	c	vertical velocity	Pa/s
138	NLEV	s	vorticity	1/s
139	1	g	surface temperature	K
140	1	g	soil wetness	m
141	1	g	snow depth (water equi.)	m
142	1	ga	large scale precipitation	m/s
143	1	ga	convective precipitation	m/s
144	1	ga	snow fall	m/s
146	1	ga	surface sensible heat flux	W/m^2
147	1	ga	surface latent heat flux	W/m^2
148	NLEV	c	horizontal streamfunktion	m^2/s
149	NLEV	c	velocity potential	m^2/s
151	1	c	mean sea level pressure	Pa
152	1	s	ln(surface pressure)	
153	NLEV	g	cloud liquid water content	kg/kg
155	NLEV	s	divergence	1/s
156	NLEV	c	geopotential height	gpm
157	NLEV	c	relative humidity	frac.
159	1	g	$(u^\star)^3$	$(m/s)^3$
160	1	ga	surface runoff	m/s

Code	Levels	Type	Variable	Unit
162	NLEV	g	cloud cover	frac.
164	1	ga	total cloud cover	frac.
169	1	ga	surface temperature	K
170	1	g	deep soil temperature	K
172	1	g	land sea mask	[0:sea,1:land]
173	1	g	surface roughness	m
175	1	g	surface albedo	frac.
176	1	ga	surface solar radiation	W/m^2
177	1	ga	surface thermal radiation	W/m^2
178	1	ga	top solar radiation	W/m^2
179	1	ga	top thermal radiation	W/m^2
180	1	ga	u-stress	Pa
181	1	ga	v-stress	Pa
182	1	ga	evaporation	m/s
183	1	g	soil temperature	K
203	1	ga	top solar radiation upward	W/m^2
204	1	ga	surface solar radiation upward	W/m^2
205	1	ga	surface thermal radiation upward	W/m^2
207	1	g	soil temperature (level 2)	K
208	1	g	soil temperature (level 3)	K
209	1	g	soil temperature (level 4)	K
210	1	g	sea ice cover	frac.
211	1	g	sea ice thickness	m
212	1	g	vegetation cover	frac.
218	1	g	snow melt (water equiv.)	m/s
221	1	g	snow depth change (water equiv.)	m/s
230	1	ga	vertical integrated spec. hum.	kg/m^2
232	1	g	glacier cover	frac.

s: PUMA spectral field; g: PUMA grid point field; c: computed by PUMA-burner; a: accumulated

Appendix C

Namelist

basic namelist parameters

Name	Def.	Type	Description
kick	1	int	0: no noise initialization (p_s = const.) 1: random white noise 2: Equator symmetric random white noise 3: mode (1,2) no random initialization
nafter	12	int	output interval for data [timesteps]
ncoeff	0	int	number of coefficients to print in **wrspam**
ndiag	12	int	output interval for diagnostics [timesteps]
nexp	0	int	experiment identifier
nkits	3	int	number of short initial timesteps
nrestart	0	int	1: preform a restart run
nrun	0	int	number of timesteps to run (excl. initial ones)
nstep	0	int	current timestep
nstop	0	int	stop at time step **nstop**
ntspd	24	int	number of timesteps per day
nrad	1	int	1: switches radiation on
nflux	1	int	1: switches vertical diffusion on
neqsig	1	int	1: use equidistant sigma levels
nprint	0	int	1: comprehensive print out (only for checks!)
nprhor	0	int	1: grid point for print out (only for checks!)
npacksp	1	int	1: pack spectral fields on output
npackgp	1	int	1: pack gridpoint fields on output
ndiaggp	0	int	1: process franks gp-diagnostic arrays
ndiagsp	0	int	1: process franks sp-diagnostic arrays
ndiagcf	0	int	1: turn on cloud forcing diagnostic
ndiaggp2d	0	int	number of additional 2-d gp-diagnostic arrays
ndiaggp3d	0	int	number of additional 3-d gp-diagnostic arrays
ndiagsp2d	0	int	number of additional 2-d sp-diagnostic arrays
ndiagsp3d	0	int	number of additional 3-d sp-diagnostic arrays
nhdiff	15	int	critical wavenumber for horizontal diffusion
ndel	NLEV*2	int	order of hyperdiffusion for each level (2*h)
ndl(NLEV)	NLEV*0	int	1: activate spectral printouts for this level
ntime	0	int	1: turn on time use diagnostics
nperpetual	0	int	radiation day for perpetual integration
tfrc(NLEV)	0,0,0,.. ,1	int	Rayleigh friction timescale τ_F in days for each level
dtns	0	int	Pole to Pole temperature difference at surface for T_R
dtep	0.0	real	Equator - Pole temperature difference at surface for T_R can simulate winter/summer hemispheres
dttrp	2.0	real	a temperature increment which controls the sharpness
dttrp			of the tropopause in T_R
dtrop	12000.0	real	height of tropopause [m]
psurf	101325.0	real	global mean surface pressure [Pa]
tdissd	0.2	real	diffusion time scale for divergence [days]
tdissz	1.1	real	diffusion time scale for vorticity [days]
tdisst	5.6	real	diffusion time scale for temperature [days]
tdissq	5.6	real	diffusion time scale for specific humidity [days]
tgr	288.0	real	global mean temperature of ground used to set T_R
time0	0.0	real	start time (for performance estimates)
restim(NLEV)	NLEV*15.0	real	restoration timescale for each level
t0(NLEV)	NLEV*250.	real	reference T_R-temperature profile

fluxmod parameters

Name	Def.	Type	Description
nevap	1	int	1: turn on surface evaporation
nshfl	1	int	1: turn on surface sensible heat flux
nstress	1	int	1: turn on surface wind stress
nvdiff	1	int	1: turn on vertical diffusion
vdiff_lamm	160.0	real	1: tuning parameter for vert. diff.
vdiff_b	5.0	real	1: tuning parameter for vert. diff.
vdiff_c	5.0	real	1: tuning parameter for vert. diff.
vdiff_d	5.0	real	1: tuning parameter for vert. diff.

miscmod parameters

Name	Def.	Type	Description
ndca	0	int	1: turn on convective adjustment

surfmod parameters

Name	Def.	Type	Description
nsurf	not active	int	debug switch
noromax	model resolution (NTRU)	int	resolution of orography
oroscale	1.0	real	scaling factor for orography

seamod parameters

Name	Def.	Type	Description
nocean	1	int	ocean model (1) or climatology (0)
nice	1	int	sea ice model (1) or climatology (0)
ncpl_ice_ocean	1	int	ice ocean coupling time steps
ncpl_ice_ocean	1	int	ice ocean coupling time steps
ncpl_atmos_ice	32	int	atmosphere ice coupling time steps
sstfile	surface_parameter	char*80	file with sst climatology
albsea	0.069	real	albedo for open water
albice	0.7	real	max. albedo for sea ice
dz0sea	$1.5 \cdot 10^{-5}$	real	roughness length sea [m]
dz0ice	$1.0 \cdot 10^{-3}$	real	roughness length ice [m]
drhssea	1.0	real	wetness factor sea
drhsice	1.0	real	wetness factor ice

icemod parameters

Name	Def.	Type	Description
ndiag	160	int	diagnostic output every **ndiag** time steps
nout	32	int	model data output every **nout** time steps
noutput	1	int	1: turn on ice model output
nflxcorr	360	int	time constant [days] for restoring
			no flux correction (=0)
			use flux correction from file (< 0)

oceanmod parameters

Name	Def.	Type	Description
ndiag	480	int	diagnostic output every **ndiag** time steps
nout	32	int	model data output every **nout** time steps
noutput	1	int	1: turn on ocean model output
nflxcorrmld	60	int	time constant [days] for restoring
			mixed layer depth (> 0), no flux correction (=0)
			use flux correction from file (< 0)
nflxcorrsst	60	int	time constant [days] for restoring
			sea surface temperature (> 0), no flux correction (=0)
			use flux correction from file (< 0)

oceanmod50 parameters

Name	Def.	Type	Description
ndiag	480	int	diagnostic output every **ndiag** time steps
nout	32	int	model data output every **nout** time steps
noutput	1	int	1: turn on ocean model output
nflxcorrsst	-1	int	flag for calculating flux correction (< 0) use flux correction from file (< 0)